ΕΛΛΗΝΙΚΟ ΜΕΣΟΓΕΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

ΣΧΟΛΗ ΜΟΥΣΙΚΗΣ ΚΑΙ ΟΠΤΟΑΚΟΥΣΤΙΚΩΝ ΤΕΧΝΟΛΟΓΙΩΝ

Τμήμα Μουσικής Τεχνολογίας και Ακουστικής

Πτυχιακή Εργασία

Σύστημα Επαύξησης Του Ήχου Της Κιθάρας Με Χρήση Της Τεχνικής Wavetable Synthesis

Guitar Audio Augmentation System With The Use Of Wavetable Synthesis Technique

Του

Ζαχαρία Ιακ. Στέλλα

Επιβλέπων καθηγητής : ***Στεφανάκης Νικόλαος***

Ρέθυμνο, Νοέμβριος 2022

Πνευματικά δικαιώματα

Η έγκριση της πτυχιακής εργασίας από το Τμήμα Μουσικής Τεχνολογίας & Ακουστικής του Ελληνικού Μεσογειακού Πανεπιστημίου δεν υποδηλώνει απαραιτήτως και αποδοχή των απόψεων του συγγραφέα εκ μέρους του Τμήματος.

Θα ήθελα να ευχαριστήσω ιδιαίτερα τον καθηγητή μου Νικόλαο Στεφανάκη , την οικογένεια μου και τους φίλους μου για την πολύτιμη συνεισφορά τους με κάθε τρόπο καθ’ όλη την διάρκεια της διεξαγωγής αυτής της πτυχιακής εργασίας.

ΠΕΡΙΛΗΨΗ

Η επαύξηση του ήχου των ακουστικών μουσικών οργάνων έχει ιδιαίτερο ενδιαφέρον σήμερα ως ερευνητικό αντικείμενο και ταυτόχρονα έχει οδηγήσει στην εμφάνιση νέων προϊόντων στην αγορά, τα οποία σχετίζονται με τη μουσική τεχνολογία. Σε αυτή την πτυχιακή εργασία, δημιουργήθηκε ένα σύστημα για την επαύξηση του ήχου της κιθάρας μέσω ενός Η.Υ.. Η επαύξηση επιτυγχάνεται με τη δημιουργία ενός παράλληλου ηχητικού στρώματος από συνθετικό ήχο, παραγόμενο με την χρήση της τεχνικής wavetable synthesis. Η διεργασία βασίζεται στην μονοφωνική μεταγραφή του παιξίματος της κιθάρας σε πραγματικό χρόνο. Η προσέγγιση παρουσιάζεται θεωρητικά, ενώ παράλληλα επεξηγείται και η υλοποίηση του συστήματος, με βάση τη γλώσσα προγραμματισμού Python. Για τον έλεγχο της εφαρμογής δημιουργήθηκε και γραφική διεπαφή χρήστη. Στην εργασία παρουσιάζονται τα αποτελέσματα της αξιολόγησης της εφαρμογής, με βάση την ακρίβεια της μουσικής μεταγραφής ενώ δίνονται κατευθύνσεις για την περαιτέρω βελτίωση της εφαρμογής.

ABSTRACT

Augmented musical instruments is not only interesting as a research trend, but has led to the appearance of new products in the market associated to music technology. In this Thesis, we present a system for augmenting the sound of a guitar by producing a synthetic sound layer using wavetable synthesis techniques. Core to the process is a real-time transcription mechanism that operates in a monophonic manner and accepts the guitar sound as input. The theory associated with the process is presented alongside with the approach that was followed for implementing the system using Python. A graphical user interface was also designed for assisting the final user. The transcription performance is evaluated using accuracy criteria and we present directions for further improvement.

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΚΑΤΑΛΟΓΟΣ ΕΞΙΣΩΣΕΩΝ

ΚΑΤΑΛΟΓΟΣ ΔΙΑΓΡΑΜΜΑΤΩΝ

Κατάλογος Πινάκων

ΣΥΝΤΟΜΟΓΡΑΦΙΕΣ

DAW	Digital Audio Workstation(Λογισμικό μουσικής επεξεργασίας)
FIR	Finite Impulse Response (Φίλτρα μη απειροστής συχνοτικής απόκρισης)
IIR	Infinite Impulse Response (Φίλτρα απειροστής συχνοτικής απόκρισης)
Rms	Root mean square
HY	Ηλεκτρονικός Υπολογιστής

ΑΠΟΔΟΣΗ ΟΡΩΝ

Data	Δεδομένα
Hardware	Υλικοτεχνικός εξοπλισμός
Wavetable	Αρχείο ήχου δεδομένων
Drivers	Λογισμικό καθοδήγησης συσκευών
Slides	Ολισθήσεις(τρόπος παιξίματος κιθάρας)
Ghost Notes	Κρυφές νότες(νότες που παίζονται πολύ γρήγορα και χωρίς ατάκα)
Chunk	Μήκος παραθύρου δεδομένων
Latency	Χρονική καθυστέρηση
Zero Crossings	Σημεία μηδενικού ακουστικού πλάτους

ΕΙΣΑΓΩΓΗ

Αναφορές για την κιθάρα υπάρχουν από τις πρώιμες εποχές, συγκεκριμένα από την εποχή της ελληνικής Αρχαιότητας, όπου η κιθάρα ανήκε στην ευρύτερη οικογένεια της λύρας. Αναμφίβολα η κιθάρα με την μορφή που την ξέρουμε σήμερα είναι ένα από τα πιο δημοφιλή μουσικά όργανα και η ύπαρξη της φημολογείται από τον 15ο αιώνα στην Ισπανία. Σήμερα, η ακουστική και η ηλεκτρική κιθάρα είναι από τα πιο δημοφιλή μουσικά όργανα. Η εταιρία Rickenbacker κατασκεύασε το 1931 την πρώτη ηλεκτροακουστική κιθάρα με κάψα. Ακολούθησε η Gibson το 1935, με το πρώτο σκάφος (hollow body) την φημισμένη ES150 που χρησιμοποιούσε τους μαγνήτες Charlie Christian. Η πρώτη ηλεκτρική κιθάρα που κατασκευάστηκε χρησιμοποιήθηκε αρχικά από τους μουσικούς της τζαζ ως ένα κούφιο όργανο, το οποίο στην συνέχεια ενισχύθηκε ηλεκτρικά από ένα σύστημα ενισχυτή - ηχείου για μεγαλύτερη ένταση και μεσουράνησε κατά την περίοδο της άνθησης του σουίνγκ. Οι πρώτες ηλεκτρικές κιθάρες διέθεταν κούφιο σώμα, ατσάλινες χορδές και ηλεκτρομαγνήτες με σπείρες από βολφράμιο, που κατασκεύαζε η εταιρία Rickenbacker το 1931. Παρόλο που μερικές από τις πρώτες κιθάρες κατασκευάστηκαν από τον Les Paul, ο πρώτος επιτυχημένος εμπορικά τύπος ηλεκτρικής κιθάρας με κούφιο σώμα ήταν το μοντέλο Fender Esquire, αρχιτεκτονικής Telecaster, της εταιρίας Fender, το 1950. Το 1954 η Fender κατασκεύασε ένα μοντέλο με τρεις μαγνήτες και με λεβιέ βιμπράτο, την περίφημη Stratocaster. Η ηλεκτρική κιθάρα στην συνέχεια αποτέλεσε καταλύτη για την ανάπτυξη πολλών μουσικών ειδών, που εμφανίστηκαν από τα τέλη του 1940 και μετά όπως τα Blues, το πρώιμο Rock N΄ Roll, το Rockabilly καθώς και το Blues Rock της δεκαετίας του 60’.

Μετά την εμπορική εγκαθίδρυση της Telecaster και Stratocaster από την Fender, τα δομικά συστατικά της ηλεκτρικής κιθάρας αυτής καθαυτής έχουν εδραιωθεί και δεν εξελίσσονται ιδιαίτερα με την πάροδο των ετών. Από εκείνα τα έτη όμως, είχε δημιουργηθεί η ανάγκη της επεξεργασίας ή επαύξησης του σήματος που λαμβάνονταν από την ηλεκτρική κιθάρα, ώστε να επιτευχθεί μερική ή πλήρη παραμόρφωση στο αρχικό σήμα. Ένας από τους πρωτοπόρους της εποχής εκείνης ήταν ο Chuck Berry, ο οποίος έσκιζε με λεπίδες τα ηχεία για να επιτύχει την παραμόρφωση που ήθελε στον ήχο του.

Με την πάροδο των ετών έχουν παρατηρηθεί σημαντικές εξελίξεις στον περιφερειακό εξοπλισμό που τροποποιεί τον ήχο της ηλεκτρικής κιθάρας. Τα πρώτα αναλογικά εφέ εμφανίστηκαν τις προηγούμενες δεκαετίες και αποτέλεσαν παγκόσμια στάνταρ για διάφορα

μουσικά είδη. Συγκρίνοντας τα με της τελευταίας γενιάς τις εφαρμογές ή τις συσκευές, οι οποίες βασίζονται στην ψηφιακή επεξεργασία σήματος, μπορούμε να πούμε ότι το συγκεκριμένο πεδίο παραμένει ανοιχτό και συνεχώς εξελίσσεται. Κάποιες από τις πιο πρωτοποριακές εφαρμογές - συσκευές που κυριαρχούν στις μέρες μας, παραθέτονται στην ενότητα 1.2.

Στη παρούσα πτυχιακή εργασία μελετάται η υλοποίηση ενός συστήματος επαύξησης του ήχου της κιθάρας σε πραγματικό χρόνο με τη χρήση Ηλεκτρονικού Υπολογιστή (ΗΥ). Η διεργασία βασίζεται στη χρήση της τεχνικής της wavetable synthesis για τη δημιουργία ρεαλιστικών ήχων από άλλα μουσικά όργανα και οι ήχοι αυτοί αξιοποιούνται για την αντικατάσταση ή τον εμπλουτισμό του φυσικού ήχου της κιθάρας. Το σύστημα σχεδιάστηκε ώστε να μην απαιτείται η χρήση επιπλέον ακουστικών αισθητήρων, πέραν του βασικού ηλεκτροακουστικού μετατροπέα με τον οποίο είναι ήδη εξοπλισμένη μία ηλεκτρική ή ηλεκτροακουστική κιθάρα. Τέλος, για τις ανάγκες της πτυχιακής εργασίας, η αξιολόγηση του συστήματος γίνεται με τη χρήση ηλεκτρικής κιθάρας, αν και αξίζει να σημειωθεί πως η όλη διεργασία θα μπορούσε κάλλιστα να υλοποιηθεί και με ακουστική/κλασσική κιθάρα.

Στις παρακάτω ενότητες, παραθέτονται αναλυτικά οι στόχοι που τέθηκαν, το απαιτούμενο θεωρητικό υπόβαθρο που χρειάζεται για την κατανόηση της συγκεκριμένης πτυχιακής εργασίας και όλη η υλοποίηση της στην γλώσσα προγραμματισμού Python. Συγκεκριμένα, στο πρώτο κεφάλαιο γίνεται μια ανάλυση των στόχων που τέθηκαν προτού υλοποιηθεί η εφαρμογή, αλλά και μια έρευνα της αγοράς των αντίστοιχων εφαρμογών που υλοποιούν την επαύξηση του σήματος της κιθάρας ή τον μετασχηματισμό τους σε πληροφορία MIDI. Στο δεύτερο κεφάλαιο, γίνεται εκτενής ανάλυση του θεωρητικού υποβάθρου που απαιτείται, όσον αφορά τις τεχνικές ανάλυσης και ανασύνθεσης του ήχου, ώστε να μπορέσουμε να επεξεργαστούμε κατάλληλα την πληροφορία που λαμβάνεται. Στο τρίτο κεφάλαιο, παραθέτετε η πλήρης υλοποίηση των διεργασιών στην Python. Στο τέταρτο κεφάλαιο, εναποθέτονται τα αποτελέσματα των πειραματικών αξιολογήσεων που διεξήχθησαν, για να έχουμε μια αντικειμενική εικόνα του κατά πόσο η εφαρμογή μας λειτουργεί σωστά. Στο πέμπτο κεφάλαιο, αναλύονται τα συμπεράσματα τα οποία αντλήθηκαν από όλη τη διεργασία. Τέλος, δίνονται πιθανές προεκτάσεις, οι οποίες θα μπορούσαν να υλοποιηθούν στο μέλλον.

1 Στόχοι εργασίας και ανασκόπηση της Αγοράς

1.1 Στόχοι Εργασίας

Προτού ξεκινήσει η υλοποίηση της όλης διεργασίας, διατέθηκε ένα χρονικό διάστημα, για να σχεδιαστεί ένα πλάνο όλων των στόχων και διεργασιών που θα επιτελούσε η εφαρμογή μας. Πρωταρχικός μας στόχος ήταν η ανάπτυξη αλγορίθμου για τον μετασχηματισμό του ήχου της κιθάρας, με βάση το ηχητικό σήμα αυτό καθ' αυτό. Δηλαδή, χωρίς την ανάγκη για περαιτέρω hardware ή την χρήση άλλης πληροφορίας, πέραν του ηχητικού σήματος.

Ως δεύτερος στόχος τέθηκε η αξιοποίηση της τεχνικής της wavetable synthesis, καθότι αυτό θα έδινε τη δυνατότητα στην εφαρμογή να επιτελεί λιγότερες διεργασίες σε πραγματικό χρόνο συγκριτικά με άλλες τεχνικές σύνθεσης ήχου. Ένας επιπλέον λόγος που επιλέχθηκε η συγκεκριμένη τεχνική σύνθεσης είναι ότι από την αρχή λειτουργίας της, θα έδινε την μέγιστη δυνατή ευελιξία στο χρήστη να μπορεί με την εισαγωγή μιας απλής ηχογράφησης να παράγει ήχους από οποιοδήποτε μουσικό όργανο ή φυσικό αντικείμενο. Συγχρόνως, επιλέχθηκε η εφαρμογή να επιτελεί μονοφωνική ανάλυση, για την επεξεργασία μικρού όγκου δεδομένων και στη συνέχεια να αξιολογηθεί για να δούμε πόσο καλά ανταποκρίνεται. Αποφασίστηκε επιπλέον ότι δεν θέλουμε να βασιστούμε στην μετατροπή του ηχητικού σήματος σε MIDI αλλά να ακολουθηθεί άλλη προσέγγιση, η οποία επιτρέπει τον συνεχή έλεγχο της διεργασίας της σύνθεσης. Επίσης, υπήρξε η σκέψη ότι θα ήταν θεμιτό το τονικό ύψος του συνθετικού ήχου που παράγεται (σήματος εξόδου), να έχει συσχέτιση με το τονικό ύψος του σήματος εισόδου, για παράδειγμα, να είναι ταυτόσημα τονικά ή να λαμβάνεται ο συνθετικός ήχος μετατοπισμένος μια οκτάβα πάνω ή κάτω, σύμφωνα με την επιθυμία του τελικού χρήστη και τις ανάγκες της εκάστοτε εκτέλεσης.

Τέλος, άλλοι δυο πολύ σημαντικοί στόχοι που τέθηκαν και εμπεριείχαν την μεγαλύτερη ανησυχία για το όλο εγχείρημα, ήταν το σύστημα να υλοποιηθεί με τέτοιο τρόπο, ώστε να είναι δυνατή η υλοποίηση των διεργασιών του, τόσο σε πραγματικό χρόνο όσο και σε περιβάλλον γραφικής διεπαφής χρήστη στην γλώσσα προγραμματισμού της Python.

1.2 Ανασκόπηση της Αγοράς

Από την εμφάνιση των πρώτων αναλογικών συστημάτων επεξεργασίας του ήχου της κιθάρας, έχουν αναδυθεί πολλές προσεγγίσεις κατά την πάροδο των ετών. Με την αξιοποίηση τους από ευρέως γνωστούς καλλιτέχνες, από διαφορετικά μουσικά είδη, έχουν καταστεί παγκόσμια στάνταρ. Τέτοιου είδους συστήματα επεξεργασίας, είναι τα μεμονωμένα εφέ, τα οποία μπορεί κάποιος να χρησιμοποιήσει κατά την εκτέλεση μιας μελωδικής γραμμής για να τροποποιήσει τον τελικό του ήχο, όπως το distortion, το reverb, το delay, το chorus, ο compressor, κλπ, αλλά και μονάδες πολυεφέ που συνδυάζουν τέτοιες λειτουργίες σε μια συσκευή.

Τα ηχητικά εφέ που παραδοσιακά εμφανίζονται στην αλυσίδα επεξεργασίας ήχου μιας ηλεκτρικής κιθάρας μπορούν να θεωρηθούν ως ένα είδος επαύξησης του ήχου και για αυτό το λόγο, η ηλεκτρική κιθάρα είναι το πιο γνωστό παράδειγμα επαυξημένου μουσικού οργάνου. Στις μέρες μας, σχεδόν όλες οι μονάδες επεξεργασίας ήχου που είχαν δημιουργηθεί στην αναλογική εποχή, έχουν καταφέρει να τροποποιηθούν σε ψηφιοποιημένη μορφή με τις ίδιες λειτουργίες. Κάτι τέτοιο δίνει την δυνατότητα αξιοποίησης των μονάδων αυτών από ψηφιακά προγράμματα μουσικής επεξεργασίας (DAW), σαν plug-in. Σαν συνέχεια αυτού, τα τελευταία χρόνια έχουν υπάρξει και προσεγγίσεις υλοποίησης πολλών εργαλείων που επιτυγχάνουν τον μετασχηματισμό του σήματος της κιθάρας σε πληροφορία MIDI τόσο σε πολυφωνική ανάλυση όσο και σε μονοφωνική. Ενδιαφέρον προς αναφορά είναι το γεγονός ότι σε μια εποχή όπου η τεχνολογία των ηλεκτρονικών υπολογιστών έχει γίνει προσιτή για τον κάθε άνθρωπο, πολλοί είναι αυτοί που αναθέτουν μέρος της ηχητικής επεξεργασίας σε ΗΥ.

Παρακάτω παραθέτονται κάποια από τα πιο πρωτοποριακά εργαλεία που υπάρχουν αυτή την στιγμή στην αγορά και επιτελούν μετασχηματισμό του σήματος ενός έγχορδου οργάνου, κυρίως της κιθάρας ή του μπάσου, σε πληροφορία MIDI.

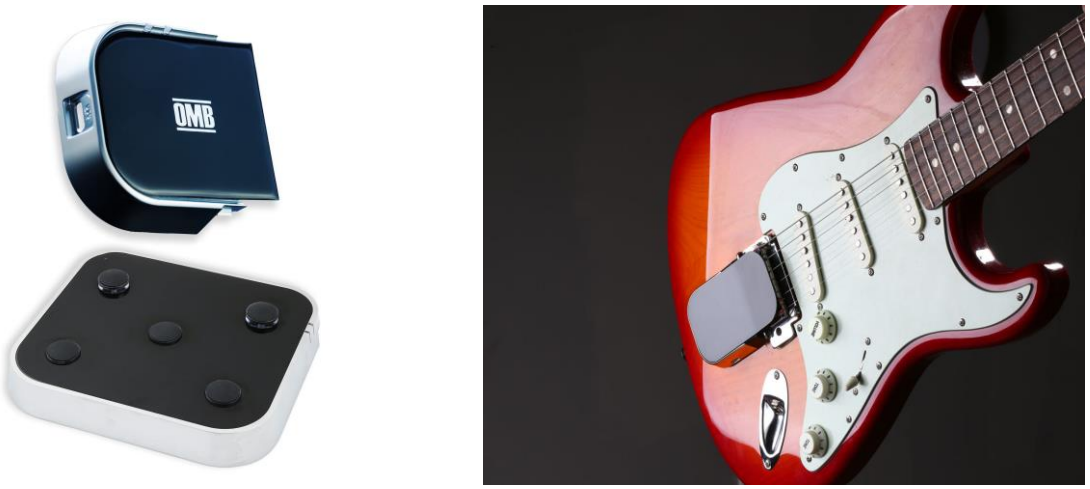
1.2.1 OMB guitar

Η OMB εισάγει μια καινούρια διάσταση στον τρόπο παιξίματος της κιθάρας με λειτουργίες όπως, της αυτόματης ενορχήστρωσης, της εναλλαγής του ήχου σε άλλων μουσικών οργάνων και της μετατροπής της πληροφορίας του σήματος της κιθάρας σε midi. Δουλεύει μετατρέποντας το σήμα που λαμβάνεται από την κιθάρα σε ηλεκτρικά σήματα που

χρησιμοποιούν χωρικά και επαγωγικά στοιχεία ανίχνευσης τα οποία αποστέλλονται στην εφαρμογή και επεξεργάζονται ώστε να γίνουν ήχοι χωρίς περαιτέρω χρονική καθυστέρηση.

Τα βασικά μέρη από τα οποία αποτελείται το OMB είναι, ο αισθητήρας που τοποθετείται πίσω από την γέφυρα, ώστε να λαμβάνεται η πληροφορία, αλλά και μια επίπεδη χάλκινη ταινία που τοποθετούμε κατά μήκος του λαιμού της κιθάρας που συνδέεται με κάθε τάστο και με τον αισθητήρα της γέφυρας. Επίσης, υπάρχει εξωτερικό χειριστήριο που μπορεί να τοποθετηθεί στο πάτωμα μπροστά από τον οργανοπαίχτη, ώστε να του δώσει την ευχέρεια να τροποποιεί τις παραμέτρους της εφαρμογής την ώρα που παίζει (Εικόνα 1).

Στην επιλογή της αυτόματης ενορχήστρωσης, το OMB δίνει την δυνατότητα να συνοδεύεται ο κιθαρίστας από ήχους παρόμοιους μιας μπάντας και οι ήχοι αυτοί μπορούν να αλλάζουν δυναμικά ανάλογα με το τι παίζει. Κάτι τέτοιο επιτυγχάνεται, μέσω του μηχανισμού αναγνώρισης συγχορδιών που έχει το OMB και προσαρμόζει ανάλογα τους υπόλοιπους ήχους, σε πραγματικό χρόνο. Εκτός της αυτόματης αναπαραγωγής, το OMB έχει άλλες δυο λειτουργίες για την εναλλαγή μουσικού οργάνου. Ο χρήστης μπορεί να επιλέξει από μια γκάμα μουσικών οργάνων και να λάβει το αντίστοιχο ηχόχρωμα της κιθάρας που επιθυμεί. Επιπλέον, παρέχεται η δυνατότητα μετατροπής της πληροφορίας σε midi για την εισαγωγή σε οποιοδήποτε λογισμικό εγγραφής μουσικού περιεχομένου, όπως το Cubase και το Ableton.



Εικόνα 1: OMB εγκέφαλος και χειριστήριο ποδιού (αριστερά) , OMB τοποθετημένο στην κιθάρα (δεξιά)

Περισσότερα για αυτό το προϊόν μπορεί κάνεις να δει στον παρακάτω σύνδεσμο και στην ιστοσελίδα της εταιρίας.

https://www.youtube.com/watch?v=gUuhxoZ06Bo&ab_channel=OMBGuitarsOfficial

https://ombguitars.com/

1.2.2 Fret Trax

Το Fret Trax έχει αναπτυχθεί από τον Lee Young. Πρόκειται για μια καινοτόμα τεχνολογία όπου τροποποιεί τον λαιμό του μπάσου με το να του προσθέσει ένα επιπλέον κομμάτι ξύλου κάτω από την επιφάνεια των τάστων. Αυτό το επιπλέον μέρος συνδέεται με το κάθε τάστο και στέλνει πληροφορία στην ηλεκτρονική πλακέτα που είναι τοποθετημένη στο πίσω μέρος του μπάσου, στην περιοχή όπου βρίσκονται τα υπόλοιπα ηλεκτρονικά στοιχεία. Για την παραμετροποίηση αυτής της νέας πληροφορίας έχουν προστεθεί στο μπροστά μέρος, στην περιοχή που βρίσκονται τα ποτενσιόμετρα, ένα επιπλέον ποτενσιόμετρο και ένα κουμπί.

Με το πάτημα του κουμπιού δίνεται η δυνατότητα να γίνει εναλλαγή στην χροιά ανάλογα με το ποιο τάστο θα πατηθεί στην υψηλότερη χορδή του οργάνου. Με το ποτενσιόμετρο μπορεί να γίνει η παραμετροποίηση της έντασης του MIDI ήχου. Αν έχει χρησιμοποιηθεί πετάλι για αυτό τον σκοπό τότε το συγκεκριμένο ποτενσιόμετρο λειτουργεί σαν pitch shift / modulation. Η καινοτομία αυτής της προσέγγισης είναι ότι εξαιτίας των δυο εξόδων υπάρχει η δυνατότητα να ακούγεται ταυτόχρονα η MIDI πληροφορία αλλά και η πληροφορία που λαμβάνουν οι μαγνήτες του μπάσου. Το συνολικό αποτέλεσμα μιας τέτοιου είδους σύνθεσης, είναι αξιόλογοι και ενδιαφέροντες ήχοι.

Εικόνα 2: Ποτενσιόμετρα (αριστερά), επιπλέον μέρος ξύλου με ηλεκτρονικά στοιχεία (κέντρο), Πλακέτα (δεξιά)

Περισσότερα για αυτό το προϊόν μπορεί κάνεις να δει στον παρακάτω σύνδεσμο και στην ιστοσελίδα της εταιρίας.

https://www.youtube.com/watch?v=eSs7acs8qZw&ab_channel=JamesRoss

https://frettrax.com

1.2.3 Fisherman Tripleplay

Αυτή η προσέγγιση είναι πολύ πιο εύκολη στην τοποθέτηση της συγκριτικά με τις προηγούμενες προσεγγίσεις, διότι για να φτάσουμε στο σημείο όπου είναι σε θέση η συσκευή να μεταδώσει πληροφορία στο λογισμικό εγγραφής, το μόνο που χρειάζεται είναι να τοποθετήσουμε τον μαγνήτη που συνδέεται με την συσκευή κοντά στην γέφυρα του οργάνου και όλη η πληροφορία θα μεταδοθεί ασύρματα στο αντίστοιχο λογισμικό.

Με αυτή την συσκευή δεν δίνεται η δυνατότητα εναλλαγής της χροιάς του ήχου με κάποιο κουμπί όμως, μπορούμε να τροποποιήσουμε οτιδήποτε χρειαστεί μέσα από το λογισμικό μουσικής παραγωγής και επεξεργασίας που χρησιμοποιούμε, ή το λογισμικό που δίνεται από την εταιρία. Δίνεται πληθώρα ήχων από βιβλιοθήκες του λογισμικού της συσκευής αλλά και από αντίστοιχα λογισμικά εγγραφής, τα οποία χρησιμοποιεί ο κάθε χρήστης. Ως αποτέλεσμα αυτών, είναι η δημιουργία νέων και ενδιαφερόντων ηχητικών συνθέσεων.

Εικόνα 3: Συσκευή αυτούσια (αριστερά), συσκευή τοποθετημένη στην κιθάρα (δεξιά).

Περισσότερα για αυτό το προϊόν μπορεί κάνεις να δει στον παρακάτω σύνδεσμο και στην ιστοσελίδα της εταιρίας.

https://www.youtube.com/watch?v=MzhORu90U0U&ab_channel=PeteThorn

https://www.fishman.com

1.2.4 Livid guitar wing

Το guitar wing είναι μια συσκευή που φτιάχτηκε για να συνδέεται με ηλεκτρικές κιθάρες και μπάσα, ώστε να παρέχει μια πληθώρα επιλογών, όσον αφορά την παραμετροποίηση του ήχου. Είναι μια συσκευή που συνδέεται ασύρματα με τον υπολογιστή και δεν χρειάζεται drivers η κάποιο άλλο λογισμικό για να είναι σε θέση να λειτουργεί.

Η πληροφορία που λαμβάνεται είναι κωδικοποιημένη μέσω του πρωτοκόλλου midi. Μέσω της συσκευής ο οργανοπαίχτης μπορεί να έχει τον έλεγχο πολλών λειτουργιών και να τις επηρεάζει όσο παίζει. Επίσης, μπορεί να χρησιμοποιήσει διάφορα εφέ τα οποία δίνονται με την συσκευή. Τέλος, μπορεί να χρησιμοποιήσει κάποιο άλλο εφέ της αρεσκείας του από αυτά που χρησιμοποιεί ή από άλλα προγράμματα μουσικής επεξεργασίας.

Η αρχή λειτουργίας του βασίζεται στην ανίχνευση της δόνησης του οργάνου από την θεμέλιο νότα της κάθε χορδής και στην συνέχεια την ανασυνθέτει αναλόγως, πράγμα που είναι καινοτόμο και δεν το έχουμε παρατηρήσει σε κάποια άλλη συσκευή μέχρι τώρα. Κάτι επίσης που αξίζει να πούμε και να συνυπολογιστεί στις καινοτομίες της συγκεκριμένης συσκευής, είναι ότι χάρη στον τρόπο σχεδιασμού της που δεν ενσωματώνεται μόνιμα επάνω στο όργανο μπορεί να χρησιμοποιηθεί και σε άλλες κιθάρες ή μπάσα με μεγάλη ευκολία.

Εικόνα 4: Το livid guitar wing κατά την διάρκεια μιας εκτέλεσης (αριστερά), το livid guitar wing τοποθετημένο (δεξιά)

Περισσότερα για αυτό το προϊόν μπορεί να δει κανείς στον παρακάτω σύνδεσμο και στην ιστοσελίδα της εταιρίας.

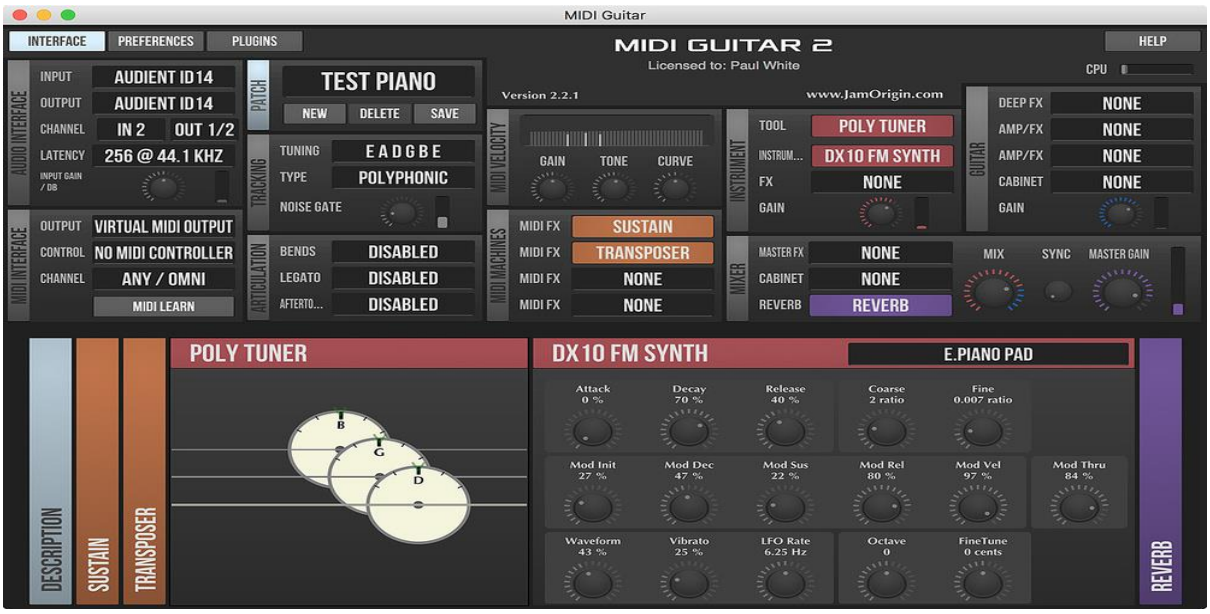
https://www.youtube.com/watch?v=9FR5ebKzf78&ab_channel=lividTV

https://guitarwing.com

1.2.5 Midi Guitar 2

Η εταιρία Jam Origin εξελίσσει το λογισμικό της στην αναγνώριση ήχων και τις τεχνολογίες μουσικής μεταγραφής από την αρχή της μέχρι και σήμερα. Το music guitar 2 είναι η πρώτη εφαρμογή πολυφωνικής μεταγραφής με πολύ χαμηλές τιμές χρονικής καθυστέρησης. Το μεγάλο πλεονέκτημα της συγκεκριμένης εφαρμογής είναι ότι πρόκειται για μια εφαρμογή αποκλειστικά λογισμικού που δουλεύει σε όλους τους τύπους κιθάρας, χωρίς δηλαδή την απαίτηση για εγκατάσταση κάποιου ειδικού hardware. Είναι σε θέση να εντοπίσει την τεχνική παιξίματος, πολύπλοκες συγχορδίες, αλλά και μονοφωνικές μελωδίες. Μπορεί και βρίσκει την αρχή (onset) και το τέλος (offset) της κάθε νότας, τις εναλλαγές νοτών χωρίς να χτυπήσουμε την χορδή (slides, ghost notes) και το πιο σημαντικό είναι ότι μπορεί και χειρίζεται τις διαφορές μεταξύ των μαγνητών της κάθε κιθάρας, τυχόν ξεκούρδιστες χορδές και τον θόρυβο των τάστων.

Παρά το γεγονός ότι εντοπίζει πολυφωνικό ήχο, αναφερόμενοι στην πολυπλοκότητα που έχει ένα τέτοιο εγχείρημα, το midi guitar 2 είναι γρηγορότερο από πολλές εφαρμογές αναγνώρισης μονοφωνικού ήχου. Αναφορικά με τον παράγοντα του latency έχει ίδιες τιμές με αντίστοιχες εφαρμογές που έχουν συνοδευτικές συσκευές, όπως αυτές που αναφέραμε παραπάνω. Αυτό που πρέπει να αναφερθεί είναι ότι, αν και θα περίμενε κανείς από μια τέτοιου είδους εφαρμογή να χρειάζεται ένα σύστημα μεγάλης υπολογιστικής ισχύος, αντ' αυτού μπορεί να λειτουργήσει ακόμα και σε ένα iPod.



Εικόνα 5: Η γραφική διεπαφή της εφαρμογής του midi guitar 2

Περισσότερα για αυτό το προϊόν μπορεί κάνεις να δει στον παρακάτω σύνδεσμο της ιστοσελίδας της εταιρίας.

https://www.jamorigin.com/

1.2.6 Sonuus i2M Musicport

Η συσκευή i2M Musicport σχεδιάστηκε με σημείο αναφοράς τις ηλεκτρικές κιθάρες και τα μπάσα. Παρ' όλα αυτά, είναι σε θέση να λειτουργήσει με τα περισσότερα μουσικά όργανα συμπεριλαμβανομένων των πνευστών και της ανθρώπινης φωνής. Μια σημαντική λειτουργία της συγκεκριμένης συσκευής είναι ότι χρησιμοποιεί τα προγράμματα καθοδήγησης συσκευής (drivers) που είναι ήδη εγκατεστημένα στο λειτουργικό σύστημα του κάθε χρήστη, πράγμα που την καθιστά πολύ γρήγορη και εύχρηστη.

Ουσιαστικά πρόκειται για μία συσκευή που υλοποιεί μονοφωνική μετατροπή του προσλαμβανόμενου σήματος του εκάστοτε οργάνου σε πληροφορία midi. Μια τέτοιου είδους διεργασία εμπεριέχει τον παράγοντα της χρονικής καθυστέρησης, όμως η τεχνολογία που χρησιμοποιείται επιτυγχάνει να κρατάει την τιμή του χαμηλή και με σωστά αποτελέσματα, όσον αφορά την εκτίμηση της νότας. Επίσης, μουσικοί παράμετροι όπως της ατάκας και του σβησίματος της νότας αποδίδονται σε πολύ ικανοποιητικό βαθμό.

Το ι2M musicport εμπεριέχει στις λειτουργίες του την διεργασία της υψηλής προενίσχυσης στο σήμα εισόδου, πράγμα που σημαίνει ότι δεν θα λαμβάνετε το σήμα

υποτονικό, κάτι το οποίο είναι συχνό πρόβλημα όταν λαμβάνετε line σήμα από κιθάρες και δυναμικά μικρόφωνα. Η εταιρία παρέχει λογισμικό που μπορεί να παραμετροποιηθεί η δρομολόγηση των καναλιών και άλλων λειτουργιών.

Εικόνα 6: Sonuus i2M Musicport

Περισσότερα για αυτό το προϊόν μπορεί κάνεις να δει στον παρακάτω σύνδεσμο και στην ιστοσελίδα της εταιρίας.

https://www.youtube.com/watch?v=QUydrqG76-E&ab_channel=ProduceLikeAPro

https://www.sonuus.com/products_i2m_mp.html#sub_menu_anchor

1.3 Wave Table Synthesis

Η Lookup table ή wavetable synthesis εφευρέθηκε το 1958 από τον Max Mathews ως ένα μέρος του λογισμικού Music II. (Cook, P. 2002). Στα μετ' έπειτα έτη γνώρισε μεγάλη απήχηση, με την άνοδο των αναλογικών και ψηφιακών synthesizers που χρησιμοποιήθηκαν κατά κόρον αλλά και σε μια πληθώρα συσκευών και εφαρμογών.

Το να οριστεί μια κυμματομορφή ενός ηχητικού σήματος με μια μαθηματική φόρμουλα δεν είναι ιδιαίτερα εύκολο πάντα. Απαιτείται αρκετή υπολογιστική ισχύ για μπορέσει να εκτελέσει όλες τις πράξεις που χρειάζονται για να δημιουργηθούν οι τιμές της κυμματομορφής και κάτι τέτοιο είναι εξαιρετικά χρονοβόρο. Για τον λόγο αυτό, οι μουσικοί και οι μηχανικοί ήχου χρησιμοποιούν σαν συνθετική πηγή έναν wavetable. Με την μέθοδο αυτή, δεσμεύεται ένας συγκεκριμένος χώρος στην μνήμη του υπολογιστή όπου περιέχει τις τιμές της κυμματομορφής. Από πλευράς υπολογιστικής ισχύος είναι αρκετά ταχύτερο συγκριτικά με το να γίνονται μαθηματικές πράξεις.

Με τον όρο wavetable αναφερόμαστε σε ένα πίνακα από N στοιχεία οπού το σύνολο τους μας δίνει το μήκος του και η κάθε τιμή του N προσδιορίζει το πλάτος της κυμματομορφής σε κάθε θέση. Υπάρχουν προσεγγίσεις όπου η εφαρμογή έχει ως δυνατότητα να μπορεί ο χρήστης να "ζωγραφίζει" την επιθυμητή κυμματομορφή ή να εισάγει όποιο αρχείο ήχου επιθυμεί να χρησιμοποιηθεί ως wavetable (Max msp). Είναι σημαντικό να αποσαφηνιστεί ότι ο wavetable δεν είναι sampler και η λειτουργία που επιτελούν αυτές οι δυο τεχνικές σύνθεσης είναι τελείως διαφορετικές. Ο wavetable είναι συνδεδεμένος με έναν point reader που ανακυκλώνει τις τιμές του με συγκεκριμένο βήμα και εξάγει την τιμή του πλάτους από την κάθε θέση διαδοχικά ώστε να δημιουργήσει την κυμματομορφή του ηχητικού σήματος. Όταν ο point reader φτάσει στην τελευταία θέση του wavetable θα επιστρέψει στην αρχή. Το εύρος του wavetable και η συχνότητα δειγματοληψίας καθορίζουν την θεμελιώδη συχνότητα του συνθετητή.

Για παράδειγμα, έστω ένας wavetable που έχει μία περίοδο ενός ημιτονικού σήματος, έστω ότι αυτό σήμα αναπαρίσταται από 1024 δείγματα και έστω ότι η συχνότητα δειγματοληψίας είναι 44100 Hz. Διαιρώντας αυτά τα δυο δεδομένα λαμβάνουμε ως αποτέλεσμα τον συνολικό χρόνο που απαιτείται για να αναπαραχθεί αυτή η συγκεκριμένη περίοδος:

$$1024 / 44100 = 0.023 \text{ sec} \quad (1)$$

Χρησιμοποιώντας την σχέση που συνδέει την περίοδο ενός ημιτονικού σήματος με τη συχνότητα του, ($f = 1 / T$) , αντικαθιστώντας το δεδομένο της διάρκειας μιας περιόδου ($T = 0.023$ sec) , λαμβάνουμε το αποτέλεσμα της συχνότητας που ηχεί και γίνεται αντιληπτή

$$f = 1 / 0.023 = 43.5 \text{ Hz} \quad (2)$$

Το βασικότερο πλεονέκτημα της τεχνικής wavetable synthesis είναι η δυνατότητα αλλαγής της τονικότητας. Για να επιτύχουμε κάτι τέτοιο, σκεπτόμενοι τις παραπάνω μαθηματικές εκφράσεις και τα δεδομένα που προέκυψαν, θα μπορούσε να πει κάνεις ότι για να υπάρξει αλλαγή στο τονικό ύψος πρέπει να υπάρξει αλλαγή είτε στο εύρος του wavetable, είτε στην συχνότητα δειγματοληψίας. Αυτές οι δυο παράμετροι δεν είναι εύκολο να τροποποιηθούν κατά την διάρκεια της σύνθεσης, ειδικότερα για προσεγγίσεις που υλοποιούνται σε πραγματικό χρόνο. Λύση στο πρόβλημα της αλλαγής τονικότητας δίνει η τροποποίηση της τιμής του βήματος του point reader που διαβάζει τον wavetable. Για

παράδειγμα, αν η τιμή του βήματος που θα διάβαζε τον wavetable ήταν 5, τότε θα λαμβάναμε ένα ημιτονικό σήμα τονικότητας ύψους 217.4 Hz.

$$(1 * 5) / 0.023 = 217.4 \text{ Hz} \quad (3)$$

Αυτό που πρέπει να λάβουμε υπόψη μας είναι ότι τροποποιώντας την τιμή του βήματος στον wavetable δημιουργούνται τιμές θέσεων με δεκαδικό μέρος που δεν υπάρχουν. Για να διαχειριστούμε αυτό το ζήτημα χρησιμοποιούμε τεχνικές που διαμορφώνουν την τιμή ανάλογα με τις κοντινότερες ακέραιες θέσεις του wavetable. Η τεχνική που χρησιμοποιήθηκε στην παρούσα πτυχιακή εργασία είναι αυτή της γραμμικής παρεμβολής (linear interpolation).

2 Περιγραφή Υλοποίησης

Για την υλοποίηση αυτής της πτυχιακής δημιουργήθηκε η ανάγκη ενός συστήματος (εφαρμογής) που θα είναι σε θέση να διαβάζει την πληροφορία που λαμβάνεται από το σήμα εισόδου (σήμα κιθάρας) πολύ γρήγορα ώστε να γίνονται όλες οι διεργασίες που επιθυμούμε για να λαμβάνουμε σήμα εξόδου σε πραγματικό χρόνο. Για μια τέτοιου είδους διεργασία είναι σωστή τακτική να επιλέξουμε πολύ μικρό μήκος παράθυρου δεδομένων (Chunk) για την επεξεργασία της πληροφορίας. Η παράμετρος της χρονικής καθυστέρησης (Latency) είναι κάτι που πρέπει να ληφθεί σοβαρά υπόψιν δεδομένου του ότι μιλάμε για μια εφαρμογή που υλοποιείται σε πραγματικό χρόνο.

Η εφαρμογή που δημιουργήσαμε έχει δυνατότητες μονοφωνικής ανάλυσης του ήχου που λαμβάνεται. Το τονικό πεδίο εφαρμογής που καλύπτει είναι από την πιο μπάσα νότα της κιθάρας E2 στα 82.41 Hz έως την G3 στα 196 Hz.

2.1 Windowing

Ο όρος window function αναφέρεται στην μαθηματική συνάρτηση όπου οι τιμές είναι μηδέν έξω από ένα συγκεκριμένο διάστημα (Cook, P. 2002). Σε πολλές εφαρμογές ανάλυσης σήματος, είναι επιθυμητή ή απαραίτητη η επεξεργασία ενός μικρού μέρους των δεδομένων για να διασφαλιστεί η σταθερότητα του συστήματος. Η λειτουργία παραθύρου είναι η πρώτη και από τις πιο σημαντικές της συγκεκριμένης πτυχιακής εργασίας γιατί διαχωρίζει την προσλαμβανομένη ροή δεδομένων από το σήμα εισόδου σε Chunks για την επεξεργασία αυτών. Συνήθως το εύρος των Chunks είναι δυνάμεις του 2 (64, 128, 256, 512, 1024). Δεδομένου ότι είναι απαραίτητο να κρατήσουμε τον παράγοντα του Latency σε χαμηλές τιμές, η τιμή του Chunk size που επιλέχθηκε ήταν αυτή των 128. Συγκεκριμένα ο χρόνος που απαιτείται για την επεξεργασία ενός παραθύρου δεδομένων (Chunk) 128 θέσεων στα 44100Hz κυμαίνεται κοντά στην τιμή των 0.003 sec. Η τιμή αυτή προσδιορίζει τον χρόνο που απαιτείται ούτως ώστε να ληφθούν οι 128 τιμές και να επεξεργαστούν. Πρέπει να λάβουμε υπόψιν επιπλέον ένα Latency των 0.003 sec αν θέλουμε να υπολογίσουμε τον χρόνο που η πληροφορία θα φτάσει στην έξοδο ήχου. Συνυπολογίζοντας και την χρονική καθυστέρηση που έχουμε από το σύστημα, μπορούμε να πούμε ότι η συνολική τιμή του Latency κυμαίνεται στα 0.01 sec.

2.2 Συχνοτική ανάλυση

Το σύστημα που υλοποιείται σε αυτή την πτυχιακή βασίζεται στη λογική ο συνθετικός ήχος που παράγεται να ακολουθεί σε πραγματικό χρόνο το τονικό ύψος της νότας της κιθάρας που παράγεται. Η συχνοτική ανάλυση είναι βασικό μέρος όλων των συστημάτων που εκτελούν μουσική μεταγραφή. Στις παρακάτω υπό-ενότητες παρουσιάζονται δύο πιθανές προσεγγίσεις για την εκτίμηση τονικότητας και αναλύονται οι λόγοι για τους οποίους η μία προσέγγιση επιλέχθηκε έναντι της άλλης.

2.2.1 Μετασχηματισμός Fourier

Ο μετασχηματισμός Fourier είναι από τους πιο διαδεδομένους μαθηματικούς μετασχηματισμούς στη ακουστική επεξεργασία σήματος, και μας επιτρέπει να αναλύσουμε ένα σήμα από το πεδίο του χρόνου, με δείκτη δείγματος *n*, στο πεδίο της συχνότητας, με δείκτη συχνότητας ω ως εξής:

$$x[n] \rightarrow X[\omega] \tag{4}$$

Ο μετασχηματισμός Fourier χρησιμοποιείται κατά κόρον στην βιβλιογραφία της μουσικής μεταγραφής (Vretblad, 2003), ωστόσο διαπιστώθηκε ότι δεν αποτελεί το ιδανικό εργαλείο για την ανάλυση της συχνοτικής πληροφορίας στα πλαίσια της παρούσας εργασίας, γεγονός που οφείλεται στο πολύ μικρό μήκος παραθύρου που επιλέξαμε για την ανάλυση (βλ. Ενότητα 2.1).

Από τον ορισμό της Αρχής της αβεβαιότητας (ή απροσδιοριστίας) (Heisenberg 1930) γνωρίζουμε ότι δεν είναι δυνατόν να έχουμε ταυτόχρονα μεγάλη ακρίβεια ανάλυσης στο χρόνο και στη συχνότητα. Πληρώνουμε δηλαδή την ακρίβεια στην πληροφορία σχετικά με το χρόνο με απροσδιοριστία ως προς τη συχνότητα. Κάτι τέτοιο μπορεί να γίνει αντιληπτό από το παρακάτω παράδειγμα που δείχνει ότι για να έχουμε ανάλυση της τάξης των 5Hz πρέπει να έχουμε ένα μεγάλο χρονικό παράθυρο:

Έστω Δf η απόσταση μεταξύ διαδοχικών συχνοτήτων κατά τη χρήση του μετασχηματισμού Fourier με μήκος N_{FFT} δειγμάτων. Η ανάλυση που μπορούμε να έχουμε στη συχνότητα για συχνότητα δειγματοληψίας *Fs* είναι

$$\Delta f = \frac{Fs}{N_{FFT}} \tag{5}$$

Παράδειγμα μεταξύ της νότας MI και της ΦΑ. Η MI έχει θεμελιώδη στα 82.4 Hz και η ΦΑ στα 87.31 Hz. Αυτό σημαίνει ότι για να βασιστούμε στο μετασχηματισμό Fourier για τη διάκριση ανάμεσα σε αυτές τις δύο νότες, η μέγιστη απόσταση που πρέπει να έχουμε μεταξύ διαδοχικών συχνοτήτων ανάλυσης είναι 87.31-82.4= 4.9 Hz. Αν υποθέσουμε ότι *Fs* = 44100 Hz, το μήκος του μετασχηματισμού Fourier που εξασφαλίζει μια τέτοια ανάλυση είναι NFFT = 9000 δείγματα, που αντιστοιχούν σε παράθυρο διάρκειας Δt=0.20 s. Αυτό το παράθυρο όμως αντιστοιχεί σε latency που είναι πολύ μεγαλύτερο από αυτό που επιτρέπεται για μία εφαρμογή που υλοποιείται σε πραγματικό χρόνο, για αυτό και δεν χρησιμοποιήθηκε η μέθοδος ανάλυσης μέσω μετασχηματισμού Fourier.

2.2.2 Φιλτράρισμα στο πεδίο του χρόνου

Η απόκριση της συχνότητας των φίλτρων αυτών εξαρτάται από τις τιμές των συντελεστών του (Orfanidis 1995). Αυτές οι τιμές είναι συνήθως δεκαδικοί αριθμοί και προσδιορίζουν τις θέσεις τους με μεγάλη ακρίβεια.

Τα **Finite Impulse Response (FIR)** ψηφιακά φίλτρα είναι αυτά που η απόκριση της συχνότητας τους είναι συγκεκριμένης διάρκειας. Η γενική μορφή της εξίσωσης που τα προσδιορίζει είναι :

$$y[n] = \sum_{k=0}^{k=M-1} b_k x(n-k) \tag{6}$$

Όπου b_k, k=1, 2, …, K είναι ο συντελεστής του φίλτρου.

Από την εξίσωση παρατηρούμε ότι η έξοδος των FIR εξαρτάται μόνο από τις προηγούμενες τιμές M που λαμβάνει στην είσοδο.

Τα FIR είναι εύκολα στην υλοποίηση τους και εγγυούνται σταθερότητα στην διεξαγωγή δεδομένων εισόδου – εξόδου. Έχουν γραμμική φάση αν σχεδιαστούν με τις σωστές παραμέτρους και χαμηλή ευαισθησία σε λάθη που προκύπτουν από την κβαντοποίηση του σήματος εισόδου.

Τα **Infinite Impulse Response (IIR)** ψηφιακά φίλτρα είναι αυτά που η απόκριση της συχνότητας είναι απειροστής διάρκειας. Η γενική μορφή της εξίσωσης που τα προσδιορίζει είναι :

$$y[n] = \sum_{k} a_k y[n-\kappa] + \sum_{\mu} b_\mu x[n-\mu] \quad (7)$$

όπου a_k, k=1, 2, …, K και b_μ μ=1, 2, …, M είναι οι συντελεστές του φίλτρου. Παρατηρούμε ότι σε αντίθεση με τα FIR φίλτρα, η έξοδος των IIR φίλτρων εξαρτάται τόσο από τις M προηγούμενες τιμές του σήματος εισόδου, όσο και από τους K προηγούμενες τιμές του σήματος εξόδου. Αυτού του είδους ο μηχανισμός χρησιμοποιείται σε κάθε υλοποίηση IIR ψηφιακών φίλτρων και είναι υπεύθυνος για την απειροστή διάρκεια απόκρισης της συχνότητας θεωρητικά. Επί του πρακτέου κάποια στιγμή σβήνει, αυτό συμβαίνει όταν οι τιμές είναι πολύ μικρές για να αναπαρασταθούν.

Τα IIR είναι χρήσιμα για υλοποιήσεις που η υψηλή ταχύτητα επεξεργασίας δεδομένων είναι σημαντικός παράγοντας διότι απαιτούν μικρότερο πλήθος συντελεστών συγκριτικά με τα FIR. Επίσης, τα IIR μπορούν να σχεδιαστούν ώστε η απόκριση της συχνότητας να είναι μια διακριτή προσομοίωση ενός αναλογικού φίλτρου.

Για τους σκοπούς της συγκεκριμένης πτυχιακής εργασίας χρησιμοποιήθηκαν IIR διάταξης ψηφιακά φίλτρα γιατί η ταχύτητα επεξεργασίας των δεδομένων ήταν σημαντικός παράγοντας, δεδομένου του ότι η εφαρμογή μας υλοποιείται σε πραγματικό χρόνο.

2.2.3 Εύρεση τονικότητας

Δημιουργούμε M peak φίλτρα με συχνοτικά κέντρα που συμπίπτουν με τη θεμελιώδη συχνότητα κάθε μίας νότας στο πεδίο ενδιαφέροντος. Το πρώτο peak φίλτρο είχε συχνοτικό κέντρο αυτό που αντιστοιχεί στην χαμηλότερη νότα MI της κιθάρας στα 82.4 Hz, το δεύτερο είχε συχνοτικό κέντρο που αντιστοιχεί στη νότα ΦΑ στα 87.31 Hz, κοκ. Η εύρεση της τονικότητας υπολογίζεται ξεχωριστά σε κάθε chunk ως ένας ακέραιος αριθμός μεταξύ 0 και M-1

$$m = argmax\ (\mathbf{E}) \quad (8)$$

όπου $\mathbf{E} = [\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_M]$ είναι το διάνυσμα με την ενέργεια του σήματος που προκύπτει από την εφαρμογή κάθε ενός peak φίλτρου. Η ενέργεια υπολογίζεται για το m-ιοστό φίλτρο ως εξής

$$\varepsilon_m = \sum_{n=1}^{N} y_m^2[n] \tag{9}$$

όπου N είναι το πλήθος δειγμάτων σε κάθε chunk.

2.3 Ηχητική Σύνθεση

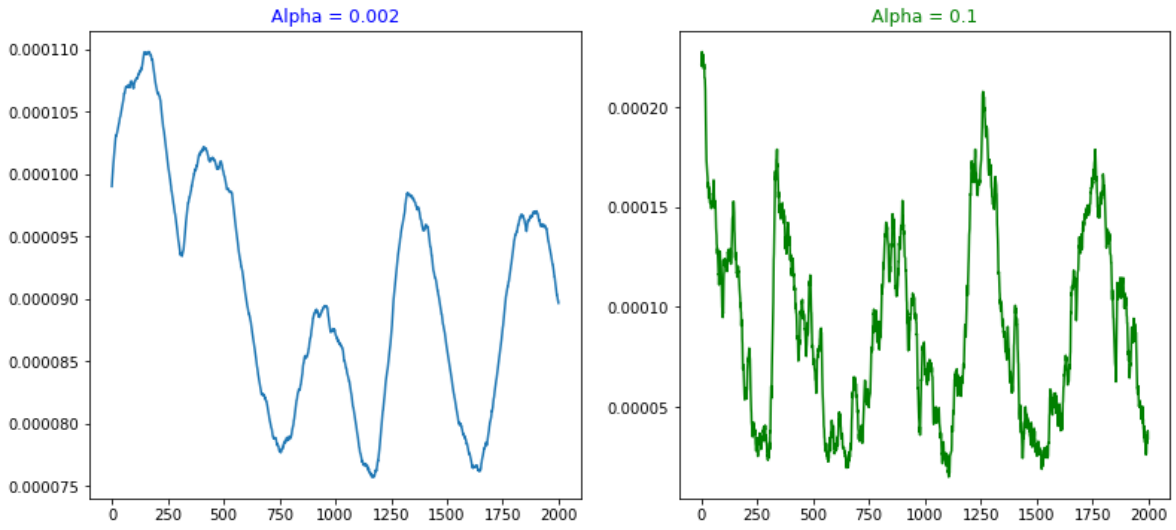
Ο τρόπος που υλοποιείται η ηχητική σύνθεση από τη συγκεκριμένη εφαρμογή έχει ως αποτέλεσμα την δημιουργία διαφορετικών και ενδιαφερόντων ήχων. Κάτι τέτοιο επιτυγχάνεται με την τεχνική της wavetable Synthesis (βλέπε ενότητα 1.3). Πέραν της διεργασίας αυτής, το σήμα μας επεξεργάζεται επιπλέον από την διεργασία του πλάτους διαμόρφωσης. Η εξίσωση που εκφράζει το πλάτος διαμόρφωσης είναι της μορφής :

$$e[n] = (1 - \alpha)e[n-1] + a|x[n]| \tag{10}$$

Ο χρηστής μπορεί να τροποποιήσει την τιμή της, α (άρα και το τελικό αποτέλεσμα του σήματος) από το slider 10 όπως φαίνεται στην εικόνα 10. Εν τέλει το τελικό σήμα που στέλνεται στην έξοδο προκύπτει από τον πολλαπλασιασμό του στιγμιαίου πλάτους με την τιμή που προκύπτει από τον wavetable

$$z[n] = e[n]w[n] \tag{11}$$

Για να μπορέσει να γίνει καλύτερη κατανόηση της επεξεργασίας που γίνεται από την διεργασία του πλάτους διαμόρφωσης, δημιουργήθηκε αρχείο κώδικα όπου του τροφοδοτούμε ένα αρχείο ήχου μιας νότας MI στα 82.4 Hz και το φιλτράρουμε από την συγκεκριμένη παράμετρο στις πιθανές ακραίες τιμές που μπορεί να λάβει. Πιο συγκεκριμένα, στην εικόνα 7 γίνεται η απεικόνιση της επίδρασης που έχει στο σήμα μας για πολύ μικρή τιμή της παραμέτρου α (α= 0.002) και για μία μέση τιμή (α = 0.1).



Εικόνα 7: Διακύμανση του πλάτους διαμόρφωσης στο χρόνο για α=0.002 (αριστερά) και α=0.1 (δεξιά)

Παρατηρώντας τα διαγράμματα διακρίνεται ότι όταν η τιμή του α λαμβάνει τιμές που προσεγγίζουν την μέγιστη, το σήμα μας έχει περισσότερες απότομες διακυμάνσεις, ενώ όταν δέχεται τιμές που είναι πολύ κοντά στο μηδέν, το σήμα μας γίνεται πιο ομαλό. Σαν μια γενικότερη παρατήρηση, μπορούμε να πούμε ότι διακρίνεται μια εξομάλυνση των διακυμάνσεων με τη μείωση της παραμέτρου α, και αυτό γίνεται αντιληπτό σαν ένας ήχος με λιγότερο περιεχόμενο στις υψηλές συχνότητες. Η παράμετρος *α* μπορεί εν τέλει να ρυθμίζεται από το χρήστη με βάση τα αισθητικά του κριτήρια.

2.4 Ανάλυση ενέργειας εγχόρδων

Για να μπορέσει να γίνει κατανόηση του σήματος που λαμβάνεται από την ηλεκτρική κιθάρα, είναι απαραίτητο να καταλάβουμε πως γίνεται η κατανομή της ενέργειας στο ακουστικό φάσμα (Γεωργάκη 2003). Σε αυτό το σημείο, πρέπει να τονίσουμε ότι η συγκεκριμένη ενότητα δεν απευθύνεται μονό στο σήμα της κιθάρας αλλά και σε σήματα που προέρχονται από οποιοδήποτε έγχορδο μουσικό όργανο.

Τα έγχορδα μουσικά όργανα παρουσιάζουν σημαντικές διαφορές στην περιβάλουσα του πλάτους σε σχέση με τα πνευστά για παράδειγμα. Συγκεκριμένα, τα έγχορδα και τα κρουστά μουσικά όργανα παρουσιάζουν ταχύ μέτωπο (attack) διότι η ένταση τους παίρνει απότομα μεγάλες τιμές και στην συνέχεια εξασθενούν πολύ γρήγορα.

Το φάσμα των συχνοτήτων ενός ήχου μπορούμε να πούμε ότι ευθύνεται στην διαφοροποίηση δυο ήχων με ιδία θεμέλιο συχνότητα (ύψους) και έντασης (ακουστότητας).

Το υποκειμενικό γνώρισμα της χροιάς των μουσικών οργάνων που αντιστοιχεί στο φάσμα συχνοτήτων αποτελείται από:

- Την θεμέλιο συχνότητα
- Τις μερικές ή παράγωγες συχνότητες της θεμελίου (partials – overtones), οι οποίες είναι:

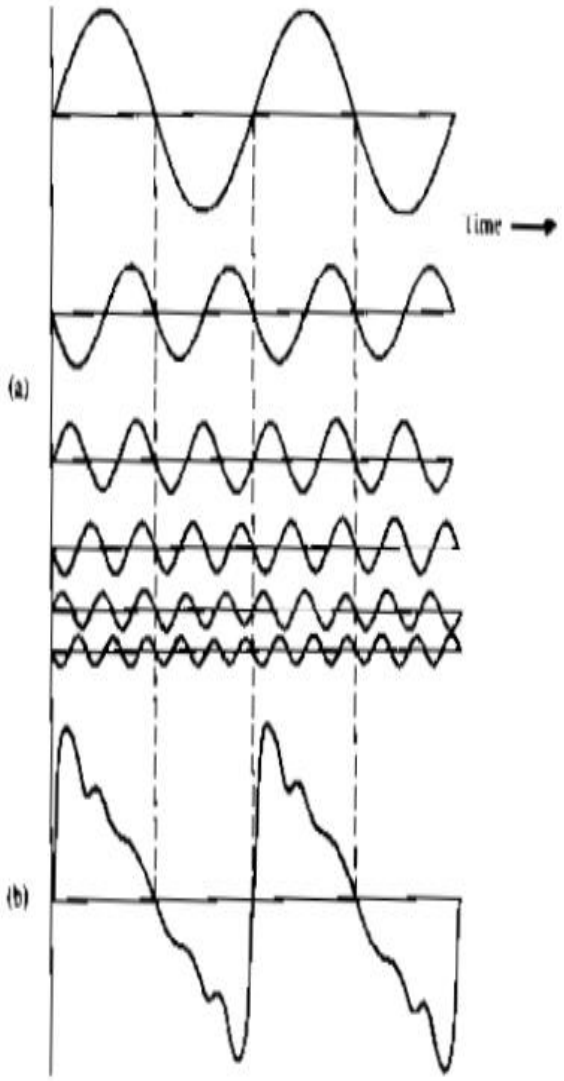
  Αρμονικοί (ακέραια πολλαπλάσια της θεμελίου)

  Μη αρμονικοί (δεκαδικά πολλαπλάσια της θεμελίου)

  Υποαρμονικοί (ακέραια υποπολλαπλάσια της θεμελίου)

  Formants (Μικρή ομάδα συχνοτήτων με κέντρο τον εκάστοτε αρμονικό)

Καταλαβαίνει κάνεις λοιπόν ότι για να μπορέσει να ταυτοποιήσει την χροιά ενός ήχου, πρέπει να αναλυθεί ο ήχος στις επιμέρους φασματικές συχνότητες που αποτελείται όπως αναφέραμε παραπάνω. Σαν ένα παράδειγμα όλων αυτών, για την καλύτερη κατανόηση τους, στην εικόνα 8 αναλύεται η νότα Σολ στα 196 Hz από ένα βιολί.



Εικόνα 8: Η σύνθετη κυμματομορφή του Σολ στα 196 Hz (b) και οι επιμέρους ημιτονοειδής κυμματομορφές που την αποτελούν(α).

3 Υλοποίηση στην Python

Η πτυχιακή εργασία που υλοποιήσαμε έχει ως στόχο την επίτευξη μετασχηματισμού του ήχου μιας ηλεκτρικής κιθάρας σε άλλων, όπως για παράδειγμα του μπάσου, σε πραγματικό χρόνο. Για να επιτευχθεί κάτι τέτοιο, η επιλογή της γλώσσας προγραμματισμού είναι καθοριστική. Η γλώσσα που υλοποιήθηκε η παρούσα εργασία είναι η Python. Ο λόγος της συγκεκριμένης επιλογής προκύπτει από την πληθώρα επιλογών και εργαλείων που μας δίνονται από την συγκεκριμένη γλώσσα προγραμματισμού.

Παρ' όλα αυτά, ενώ άλλες γλώσσες όπως για παράδειγμα η C++ έχουν μεγαλύτερες δυνατότητες από πλευράς ταχύτητας επεξεργασίας δεδομένων, η ευχρηστία που προσφέρει η Python σε βιβλιοθήκες, επικοινωνία με άλλες συσκευές και τρόπο γραφής είναι κάτι που είναι πιο δύσκολα επιτεύξιμο σε μια γλώσσα όπως η C++. Γλώσσες επιστημονικού προγραμματισμού όπως η MATLAB δεν ενδείκνυνται για το εύρος μια τέτοιας εργασίας για λόγους υπολογιστικής ταχύτητας και προσβασιμότητας συσκευών.

Από την στιγμή που πάρθηκε η απόφαση να λειτουργεί η εφαρμογή μας στα πλαίσια γραφικής διεπαφής χρήστη, ήταν επόμενο η δομή της αρχιτεκτονικής της να μοιραστεί σε δυο μέρη. Το Back-end μέρος όπου είναι υπεύθυνο για όλες τις διεργασίες που επιτελούνται στο παρασκήνιο και το Front-end μέρος που είναι υπεύθυνο για την σωστή αλληλεπίδραση με τον χρήστη.

3.1 Back-end

Σε αυτό το κομμάτι επεξεργασίας επιτελούνται όλες οι διεργασίες που γίνονται στο υπόβαθρο της εφαρμογής μας και δεν είναι ορατές από τον χρήστη. Τέτοιου είδους διεργασίες είναι, αρχικά, η λήψη και επεξεργασία ανά chunk σήματος από την ηλεκτρική κιθάρα. Στην συνέχεια, το κάθε chunk πληροφορίας επεξεργάζεται με την διαδικασία που αναφέραμε στις υπό-ενότητες 2.1, 2.2.2. Αφού γίνει η εκτίμηση της νότας , η πληροφορία επεξεργάζεται περαιτέρω με την τεχνική της wavetable synthesis όπως αναφέραμε στην ενότητα 1.3 και παραμετροποιείται περαιτέρω από το πλάτος διαμόρφωσης, υπό-ενότητα 2.3. Ως τελευταία διεργασία που επιτελείται από το back-end μέρος της εφαρμογής μας, είναι η αποστολή της τελικής μίξης της πληροφορίας του μετασχηματισμένου ήχου στην έξοδο ήχου για την αναπαραγωγή του.

3.1.1 Περιβάλλον και Βιβλιοθήκες

Για να μην υπάρξουν προβλήματα σταθερότητας του λογισμικού περιβάλλοντος και κακή διανομή των βιβλιοθηκών που επρόκειτο να χρησιμοποιηθούν, επιλέχθηκε το περιβάλλον λογισμικού της Anaconda. Ο λόγος που επιλέχθηκε είναι για την σταθερότητα που παρέχει και την ευκολία της παραμετροποίησης των λειτουργιών που παρέχονται. Ως editor χρησιμοποιήθηκε το περιβάλλον της Spyder. Οι βιβλιοθήκες που χρησιμοποιήθηκαν για την επίτευξη του στόχου που έχουμε θέσει στην παρούσα πτυχιακή εργασία, είναι :

- **NumPy**

  Η NumPy βιβλιοθήκη είναι από τις πιο κλασικές στην python και φημίζεται για την ταχύτητα εκτέλεσης των πράξεων. Μας βοηθάει να κάνουμε τους απαραίτητους μαθηματικούς υπολογισμούς και μας παρέχει τις βασικές δομές δεδομένων.

- **PyAudio**

  Η PyAudio είναι η σημαντικότερη βιβλιοθήκη για την επίτευξη της συγκεκριμένης πτυχιακής εργασίας. Η PyAudio μας παρέχει την λήψη την ηχητικών συμβάντων αλλά και της δρομολόγησης αυτών στα ηχεία.

- **SciPy**

  Η SciPy είναι μια εξίσου πολύ σημαντική βιβλιοθήκη διότι παρέχει όλες τις απαραίτητες συναρτήσεις για την επεξεργασία και το φιλτράρισμα του σήματος που λαμβάνεται. Επίσης παρέχει συναρτήσεις για την εισαγωγή αρχείων .wav, κάτι που χρησιμοποιούμε για τα αρχεία του wavetable.

- **Built-In**

  Από built-in βιβλιοθήκες χρησιμοποιήθηκαν η **Tkinter**, η **Time** και η **OS.**

- **Librosa**

  Η librosa είναι βιβλιοθήκη για ανάλυση του ήχου και της μουσικής. Χρησιμοποιήθηκε για την ανάλυση πληροφορίας αρχείων wav σε κατάλληλα format.

- **Tgt**

  Η βιβλιοθήκη tgt έχει δημιουργηθεί για την ανάγνωση και ανάλυση της πληροφορίας των textgrid αρχείων που δημιουργούνται από την επισημείωση στην Praat.

3.1.2 Εκτίμηση τονικότητας

Όπως αναφέραμε εκτενέστερα στην υπό-ενότητα 2.2.2 για το πως δημιουργείται πληροφορία για την εκτίμησης της τονικότητας (μέγιστης ενέργειας φίλτρου). Ουσιαστικά, χρησιμοποιούμε την πληροφορία των συντελεστών που δημιουργήθηκαν από την συνάρτηση

```
B, A = signal.iirpeak(freq, Q[idx], int(samplerate))
```
(122)

για να λάβουμε τις παραμέτρους A, B , οι οποίοι μας προσδιορίζουν τις απαιτήσεις του κάθε συχνοτικού κέντρου και επιστρέφουν ένα μονοδιάστατο πίνακα τριών τιμών έκαστος (Bressert 2012). Στη συνέχεια, αντικαθιστώντας τις παραμέτρους (A(As) , B(Bs)) στην συνάρτηση

```
filtsig, ic= signal.lfilter(Bs[i], As[i], d, zi=IC[:,i])
```
(133)

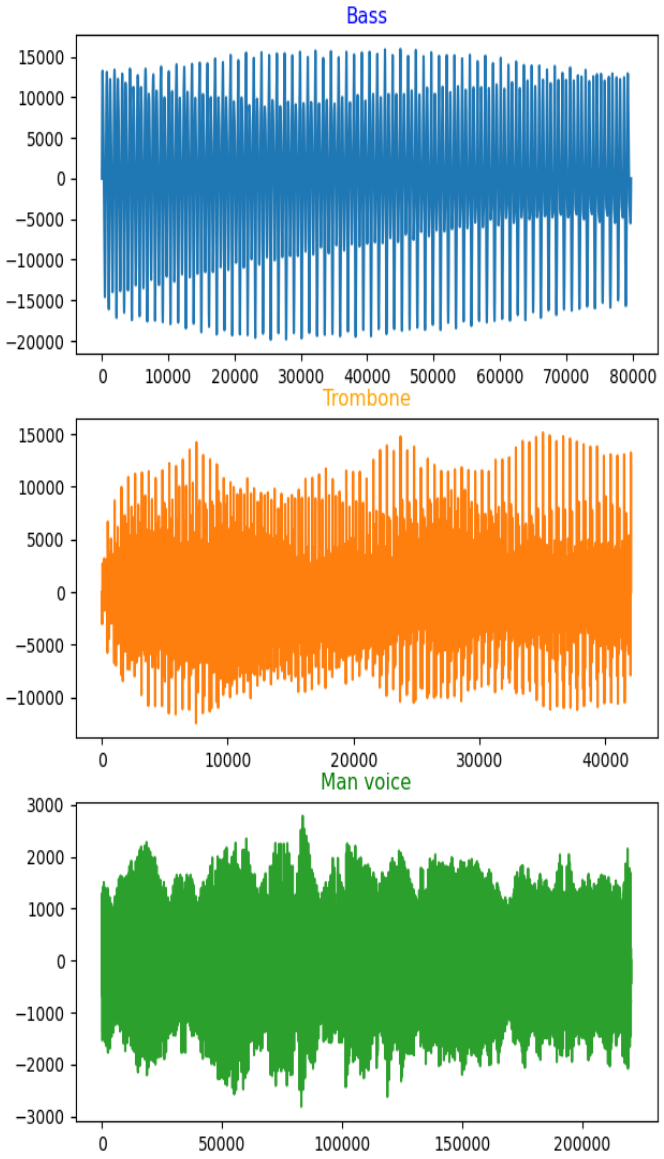
που μας δίνεται από την SciPy, λαμβάνουμε μια φιλτραρισμένη εκδοχή του κάθε chunk του προσλαμβανόμενου σήματος ανάλογα με την πληροφορία που έχει δημιουργηθεί από το κάθε IIR peak φίλτρο για το κάθε συχνοτικό κέντρο. Από αυτή την εκδοχή υπολογίζουμε την RMS τιμή με τη χρήση της ευκλείδειας νόρμας που μας δίνετε από την συνάρτηση

```
rmsCurrentFrame[0, i] = np.linalg.norm(filtsig)
```
(144)

3.1.3 Wavetables

Ως συνέχεια του τρόπου λειτουργίας της wavetable Synthesis που αναφέραμε εκτενέστερα και στο κεφάλαιο 1.3, στο κεφάλαιο αυτό παραθέτονται επιπλέον πληροφορίες για το θέμα αυτό και συγκεκριμένα για το μέρος των wavetable και το τι προϋποθέσεις πρέπει να πληρούνται ώστε να έχουμε σωστά αποτελέσματα (Roads1996).

Θεωρητικά ως wavetable μπορεί χρησιμοποιηθεί οποιοσδήποτε ήχος από την στιγμή που τον έχουμε την κατοχή μας σε ψηφιακή μορφή. Κάτι τέτοιο είναι ιδιαίτερα χρήσιμο διότι δεν μας περιορίζει ως προς το ποιο αρχείο ήχου θα χρησιμοποιήσουμε, αλλά πρέπει να προσέξουμε ώστε κάποιες προϋποθέσεις να τηρούνται για να έχουμε μια επιτυχημένη λειτουργία της wavetable synthesis. Τέτοιου είδους προϋποθέσεις είναι:



Εικόνα 9: Wavetables που χρησιμοποιήθηκαν για το μετασχηματισμό του ήχου της κιθάρας, ήχος μπάσου (πάνω), ήχος από τρομπόνι (μέση) και ήχος ανθρώπινης φωνής (κάτω). Όλα τα δείγματα είναι σε συγκεκριμένο τονικό ύψος.

- Να φροντίσουμε ώστε το αρχείο μας να είναι κοντά σε σημείο μηδενικού πλάτους. Είναι σημαντικό στο αρχείο μας να μην δημιουργούνται ασυνέχειες που να γίνονται αντιληπτές. Μια καλή πρακτική για την αποφυγή ενός τέτοιου φαινομένου είναι να φροντίζουμε η πρώτη και η τελευταία τιμή να περνάει από το μηδέν (zero crossings).
- Να έχει συγκεκριμένη τονικότητα σε όλο το εύρος του. (Η πληροφορία της τιμής της τονικότητας χρησιμοποιείται ως παράγοντας παραμετροποίησης του βήματος όπως

αναφέραμε στην ενότητα 1.3, άρα και της σωστής λειτουργίας του (pitch modulation)

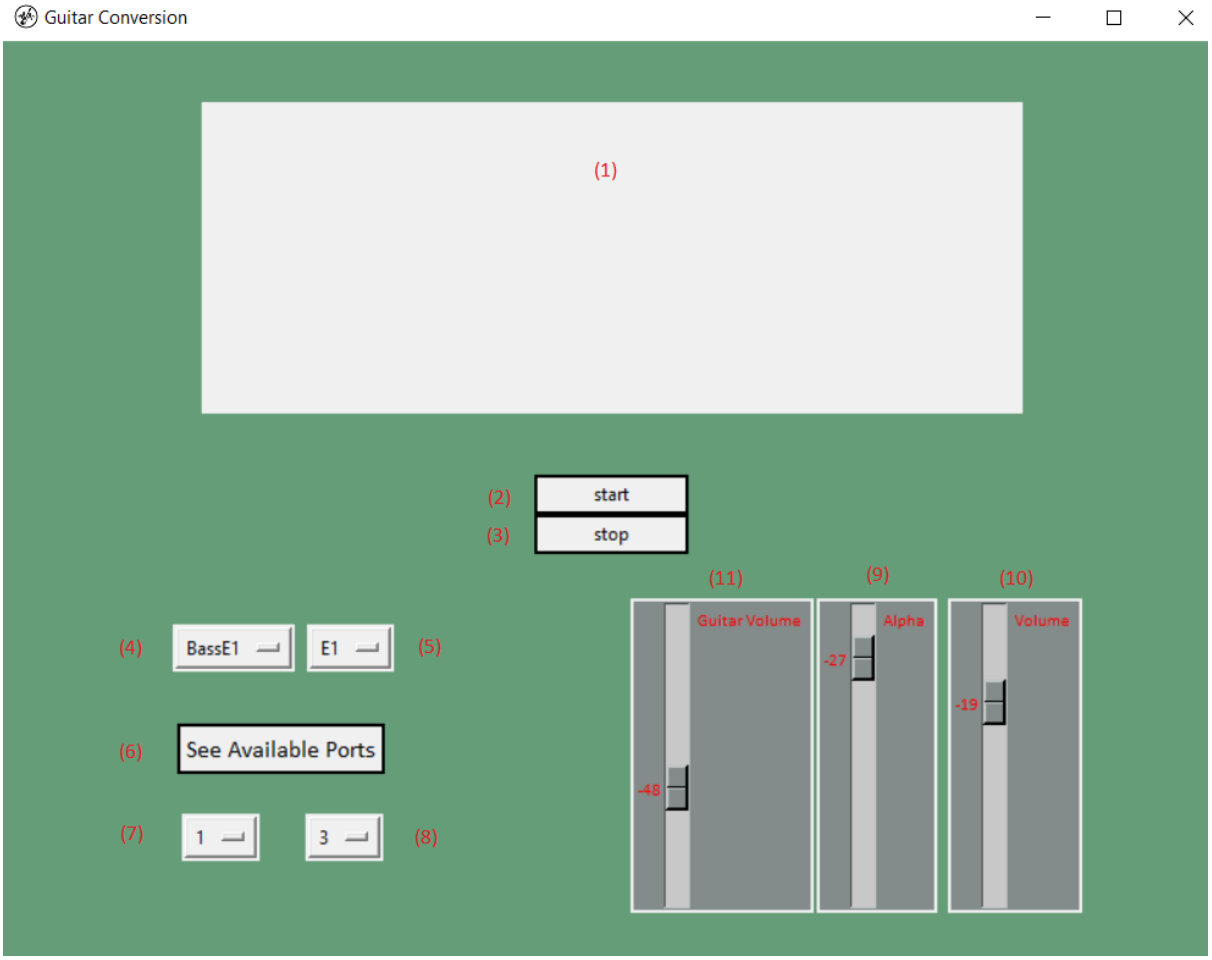
Στην συγκεκριμένη πτυχιακή εργασία χρησιμοποιήθηκαν τρία wavetables για την μετατροπή του σήματος της ηλεκτρικής κιθάρας. Αυτά τα αρχεία ήχου προσδιορίζουν έναν ήχο από ηλεκτρικό μπάσο τονικότητας E1 στα 41.2Hz, έναν ήχο από τρομπόνι τονικότητας E2 στα 82.4Hz και έναν ήχο από ανδρική φωνή τονικότητας B♭ στα 58Hz όπως παραθέτονται παραπάνω (εικόνα 9).

3.2 Front-end

Το συγκεκριμένο τμήμα επεξεργασίας ευθύνεται για όλες τις διεργασίες που είναι ορατές από τον χρήστη και μπορεί να αλληλεπιδρά μαζί τους ώστε να ρυθμίζει τις παραμέτρους που χρειάζονται για να λειτουργεί σωστά η εφαρμογή. Αυτές οι παράμετροι απευθύνονται στις λειτουργίες που επιτελούνται στο παρασκήνιο (Back-end) όπως αναφέραμε στην προηγούμενη ενότητα 3.1. Τέτοιου είδους παράμετροι είναι το ξεκίνημα και σταμάτημα της εφαρμογής, η επιλογή του επιθυμητού wavetable και άλλες αντίστοιχες λειτουργίες που θα αναλυθούν εκτενέστερα στην επόμενη υπό-ενότητα 3.2.1.

3.2.1 Εικονική Διεπαφή Χρήστη

Η εικονική διεπαφή χρήστη δημιουργήθηκε την δεκαετία του 80 από την Xerox. Πραγματεύεται ένα γραφικό περιβάλλον που δημιουργούμε γραφικά αντικείμενα ούτως ώστε ο χρήστης να δίνει εντολές ή να λαμβάνει δεδομένα πατώντας τα. Για το τμήμα που ο αλγόριθμος μας, πήρε την μορφή γραφικού περιβάλλοντος με τον χρήστη, χρησιμοποιήθηκε η built-in βιβλιοθήκη της python, **Tkinter**. Μέσω αυτής της βιβλιοθήκης κατέστη δυνατό να δημιουργηθούν κουμπιά και faders για την διαχείριση της εφαρμογής. Στις επόμενες εικόνες 10 και 11, μπορεί κάνεις να κάνει μια πιο λεπτομερή αντιστοίχιση του κάθε κουμπιού ή fader συσχετίζοντας την λειτουργία που επιτελεί.



Εικόνα 10: GUI εφαρμογής

Στην εικόνα 10 παρατίθεται η βασική μορφή που έχει η εφαρμογή μας κάθε φορά που την εκκινούμε από τον compiler. Συγκεκριμένα στην εικόνα 10 έχει αριθμηθεί κάθε λειτουργία που επιτελείται και θα σχολιαστεί παρακάτω.

1) Πρόκειται για το κεντρικό παράθυρο που μας εμφανίζει την νότα που αντιστοιχεί στην τιμή που λήφθηκε από την διεργασία της εκτίμησης νότας. Η τιμή της νότας που εμφανίζει ανανεώνεται κάθε φορά που δημιουργείται η απαραίτητη πληροφορία από το κάθε Chunk.

2) Το κουμπί που είναι υπεύθυνο για την ενεργοποίηση της εφαρμογής μας.

3) Το κουμπί που είναι υπεύθυνο για προσωρινή διακοπή της εφαρμογής μας.

4) Η λίστα που μπορούμε να επιλέξουμε το επιθυμητό wavetable. Μπορούμε να επιλέξουμε μια από τις επιλογές που αναφέραμε στην ενότητα 3.1.3 ή να χρησιμοποιήσει ο χρήστης κάποιον wavetable της επιλογής του.

5) Η λίστα για την επιλογή της κεντρικής συχνότητας του wavetable που έχουμε επιλέξει.(Πολύ σημαντική παράμετρος για την σωστή λειτουργία της wavetable synthesis αλλά και της εφαρμογής γενικότερα)

6) Πρόκειται για το κουμπί που μας εμφανίζει τις πιθανές εισόδους και εξόδους που μπορούμε να χρησιμοποιήσουμε στο εκάστοτε σύστημα που τρέχει η εφαρμογή. Η συγκεκριμένη λειτουργία αναλύεται περισσότερο παρακάτω στην εικόνα 11.

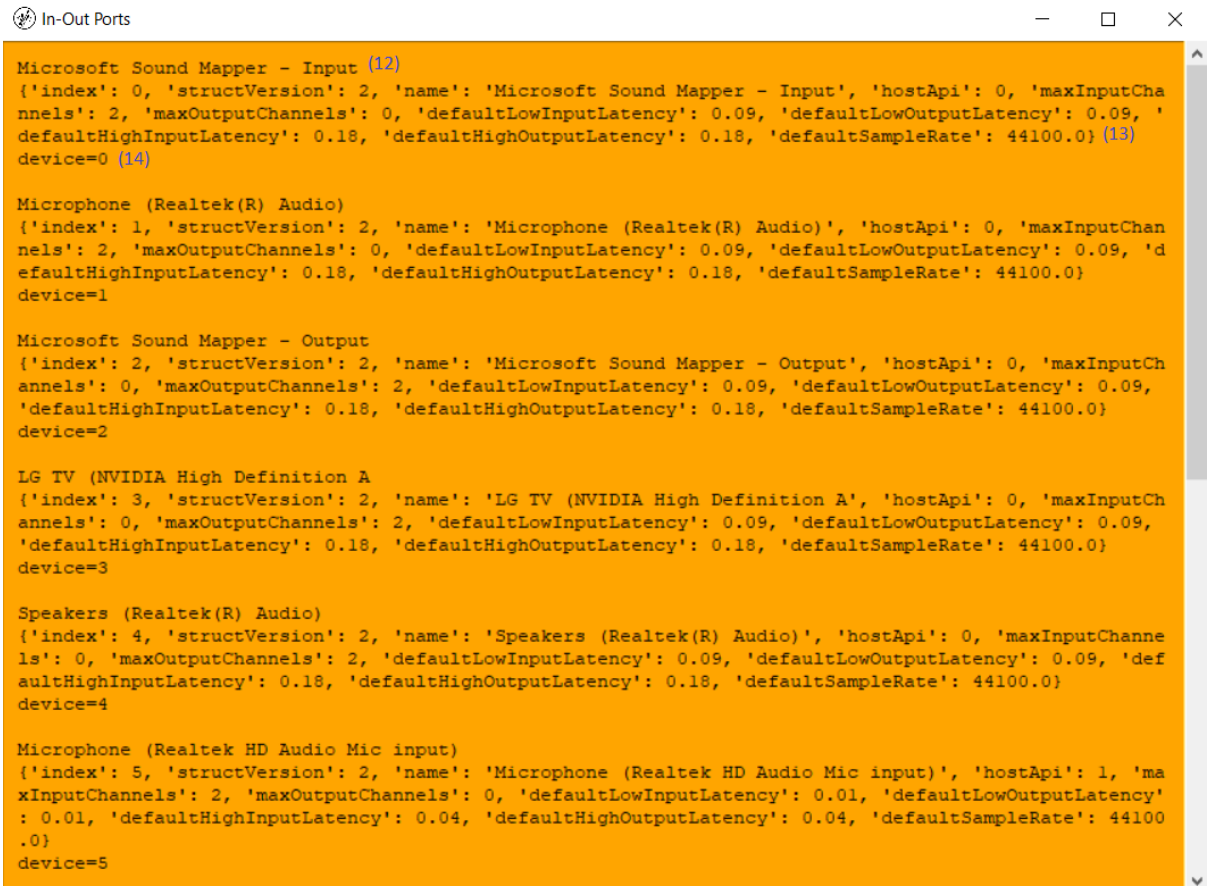
7) Η λίστα που ο χρήστης μπορεί να επιλέξει την τιμή του index που αντιστοιχεί στην είσοδο ήχου της επιλογής του.

8) Η λίστα που ο χρήστης μπορεί να επιλέξει την τιμή του index που αντιστοιχεί στην έξοδο ήχου της επιλογής του.

9) Slider που παραμετροποιεί την τιμή του α του πλάτους διαμόρφωσης που αναλύσαμε στην ενότητα 2.3.

10) Slider που παραμετροποιεί την εξαγόμενη ένταση του συνθετικού ήχου που λαμβάνουμε από τα ηχεία μας.

11) Slider που παραμετροποιεί την εξαγόμενη ένταση του φυσικού ήχου που λαμβάνουμε από την κιθάρα χωρίς περαιτέρω επεξεργασία από την εφαρμογή.



Εικόνα 11: GUI In-Out Ports

Στην εικόνα 11 παρουσιάζεται το δεύτερο παράθυρο της γραφικής εφαρμογής όπου μας παραθέτει μια λίστα με όλες τις πιθανές εισόδους και εξόδους που μπορούμε να έχουμε στην διάθεση μας ανάλογα με το σύστημα και τις συνδεδεμένες συσκευές που έχουμε στην διάθεση μας. Η συγκεκριμένη λειτουργία εκκινείτε από το κουμπί με τον αριθμό (6) όπως είδαμε στο σχολιασμό της εικόνας 10.

Αναλυτικότερα για το παράθυρο (εικόνα 11) που απαρτίζονται οι είσοδοι και οι έξοδοι ήχου της εφαρμογής μας έχουμε:

12) Το όνομα της συσκευής.

13) Τα στοιχεία της συσκευής.

14) Την τιμή του index της εκάστοτε συσκευής.

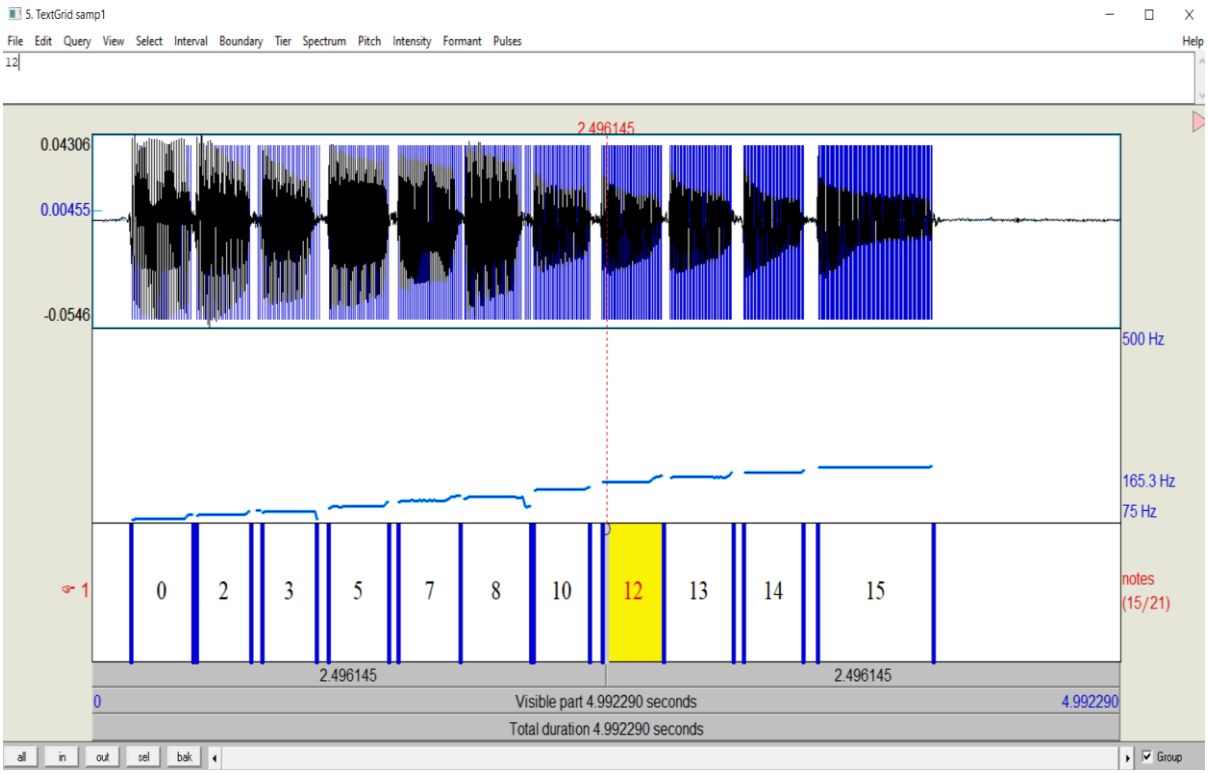
Η τιμή index (14) χρησιμοποιείται για να ρυθμίσουμε τις παραμέτρους των εισόδων και εξόδων ήχου της εφαρμογής(αναλύονται εκτενέστερα στον σχολιασμό της εικόνας 10, (7) και (8)).

4 Πειραματική αξιολόγηση

4.1 Ακρίβεια της εκτίμησης του τονικού ύψους

Για να μπορέσουμε να έχουμε αντικειμενικά αποτελέσματα του κατά πόσο η εφαρμογή παράγει σωστά αποτελέσματα έπρεπε να δημιουργηθεί μια μεθοδολογία η οποία να παράγει αντίστοιχα αποτελέσματα. Για την επίτευξη του στόχου αυτού χρησιμοποιήθηκε η τεχνική της Ground Truth όπως αποκαλείται στην ανάλυση σήματος. Ουσιαστικά ο όρος Ground Truth αναφέρεται σε ένα αρχείο που έχουμε δημιουργήσει εμείς και έχουμε σημειώσει με ακρίβεια την αρχή και το τέλος κάθε νότας αλλά και το όνομα της.

Κάτι τέτοιο κατέστη δυνατό από την εφαρμογή της Praat, όπου μας έδωσε την δυνατότητα μέσω των εργαλείων της να σημειώσουμε λεπτομερώς όλη την απαραίτητη πληροφορία που χρειαζόταν και να την εξάγουμε σε ένα αρχείο κειμένου, εικόνα 12.



Εικόνα 12: Επισημείωση ηχογράφησης με βάση το τονικό ύψος στην Praat

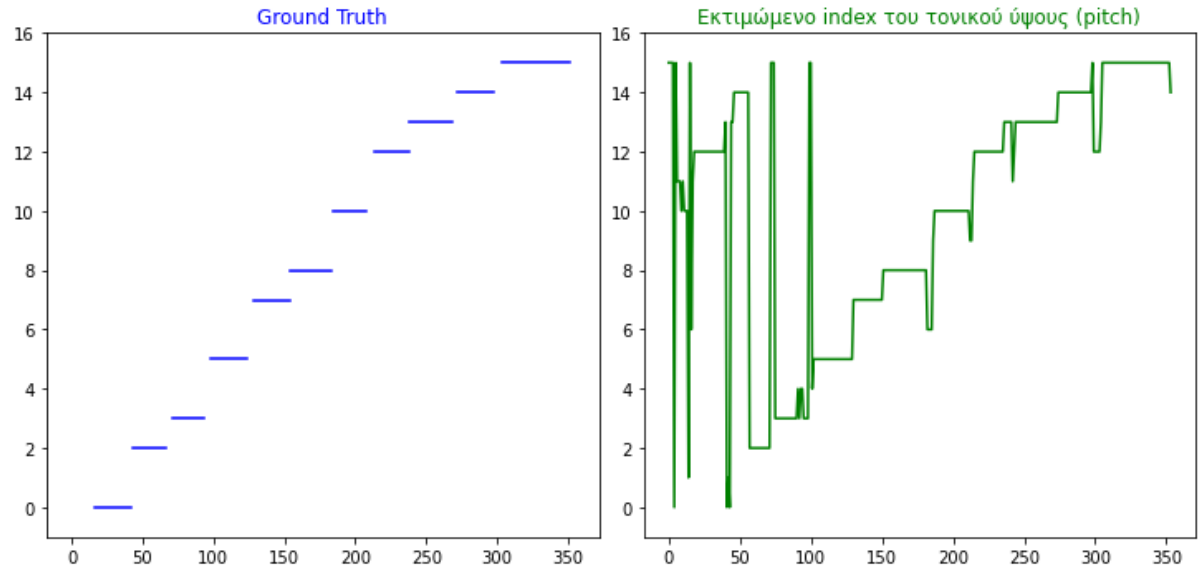
Χρησιμοποιώντας την βιβλιοθήκη tgt μπορέσαμε να δημιουργήσουμε αρχείο κώδικα ώστε να λάβουμε στην πληροφορία και να την επεξεργαστούμε κατάλληλα για να ταυτοποιήσουμε το ποσοστό επιτυχίας του συστήματος μας. Δημιουργήθηκαν τρία αρχεία

ήχου ώστε να έχουμε μια αντικειμενική εικόνα της απόδοσης του συστήματος μας. Συγκεκριμένα, ένα από τα αρχεία αναφέρεται σε ένα riff από τον χώρο της rock, ένα άλλο αναφέρεται σε μια κλίμακα που καλύπτει το εύρος των πιθανών τονικοτήτων και το τελευταίο αναφέρεται σε γρήγορες εναλλαγές μεταξύ των νοτών με διαφορετική ρυθμική αγωγή. Στον παρακάτω πίνακα συγκεντρώνεται το ποσοστό της επιτυχημένης ανάλυσης του συστήματος μας για το κάθε αρχείο ήχου, αλλά και ο συνολικός μέσος όρος αυτών.

Riff από τον χώρο της rock	84.68 %
Κλίμακα	77.14 %
Διαφορετική ρυθμική αγωγή	95.74 %
Μέσος όρος	85.85%

Πίνακας 1: Ακρίβεια εκτίμησης τονικού ύψους για κάθε ηχογράφηση

Πέραν της ποσοστιαίας ανάλυσης του κάθε αρχείου ήχου, στην εικόνα 13, εξάγεται και πληροφορία γραφικής απεικόνισης και ταυτοποίησης του κάθε αρχείου ήχου για την καλύτερη κατανόηση της συμπεριφοράς του συστήματος μας, αλλά και του εντοπισμού των σημείων που η εφαρμογή μας δεν ανταπεξήλθε όπως αναμενόταν.



Εικόνα 13: Ground Truth του τονικού ύψους (αριστερά) και εκτιμώμενο τονικό ύψος (δεξιά).

Παρατηρώντας τα διαγράμματα, εκ πρώτης όψεως διακρίνουμε κυρίως στην αρχή του διαγράμματος μια τυχαιότητα στην εκτίμηση νότας με πολύ γρήγορες διακυμάνσεις στο τονικό ύψος. Μια πρώτη υπόθεση που θα μπορούσε να κάνει κάποιος είναι ότι οφείλεται σε παρουσία θορύβου, παρατηρώντας όμως καλύτερα διαπιστώνεται ότι πρόκειται για λάθη οκτάβας. Πιο συγκεκριμένα, βλέπουμε ότι όταν παίζονται οι πιο μπάσες νότες, το σύστημα μας ταυτοποιεί την πραγματική νότα με την πρώτη αρμονική της (μια οκτάβα ψηλότερα). Ο λόγος που λαμβάνεται μια κακή εκτίμηση όσον αφορά τις μπάσες νότες, είναι ότι οι μπάσες νότες έχουν αρμονικές που συμπίπτουν με τις κεντρικές συχνότητες κάποιων εκ των φίλτρων ανάλυσης που έχουμε. Οι πρώτες αυτές αρμονικές φαίνεται λοιπόν να έχουν ενέργεια που κατά περιόδους ξεπερνάει την ενέργεια της θεμελιώδους, με αποτέλεσμα το σύστημα να μπερδεύεται. Ένας επιπλέον λόγος που συντελεί σε αυτό είναι η πολύ μεγάλη περίοδος που αντιστοιχεί στη θεμελιώδη νότα. Η περίοδος για παράδειγμα της νότας MI στα 84.41 Hz είναι μεγαλύτερη από το chunk που έχουμε ορίσει, άρα έχουμε ένα πολύ μικρό παράθυρο για να ανιχνεύσουμε ένα περιοδικό φαινόμενο που χρειάζεται μεγαλύτερο διάστημα παρατήρησης. Με αυτή την συνθήκη θα μπορούσε να σκεφτεί κανείς ότι η εφαρμογή μας δεν θα είναι σε θέση να λειτουργήσει, όμως τα φίλτρα που χρησιμοποιούμε έχουν κάποιο είδος μνήμης η οποία κρατάει πληροφορία από το προηγούμενο chunk και δουλεύει.

5 Συμπεράσματα και προτάσεις για μελλοντική εργασία

5.1 Συμπεράσματα

Σκεπτόμενοι τον αρχικό μας στόχο, της δημιουργίας μιας εφαρμογής που να είναι σε θέση να μπορεί να μετασχηματίζει τον ήχο μιας ηλεκτρικής κιθάρας σε οποιονδήποτε άλλο επιλέξει ο χρήστης, είμαστε σε θέση να πούμε ότι αυτό επετεύχθη σε πολύ καλό βαθμό. Η γλώσσα Python, παρόλο που δεν είναι η βέλτιστη επιλογή για εφαρμογές πραγματικού χρόνου, είδαμε ότι αντεπεξήλθε σε πολύ καλό βαθμό, ακόμα και με την εφαρμογή ενός μικρού chunk size της τάξης των 128 δειγμάτων, το οποίο ήταν καθοριστικής σημασίας για την μείωση του latency.

Το σύστημα υλοποιήθηκε με βάση τη λογική ο συνθετικός ήχος που παράγεται να ακολουθεί σε πραγματικό χρόνο το τονικό ύψος της νότας της κιθάρας που παράγεται. Καθοριστική για το αυτό το κομμάτι ήταν η χρήση μιας συστοιχίας ζωνοπερατών IIR φίλτρων, με συχνοτικά κέντρα που συμπίπτουν με τις θεμέλιες συχνότητες κάθε νότας. Τα φίλτρα υλοποιήθηκαν στο πεδίο του χρόνου και η εκτίμηση της τονικότητας βασίστηκε στο κριτήριο της μέγιστης ενέργειας, το οποίο έχει την ιδιότητα ότι υπολογίζεται με πολύ χαμηλή πολυπλοκότητα. Η συγκεκριμένη προσέγγιση μπορούμε να πούμε ότι δουλεύει, αλλά όχι στον επιθυμητό βαθμό, οπότε σίγουρα εδώ θα άξιζε τον κόπο να ερευνηθούν και διαφορετικές προσεγγίσεις που να βελτιώνουν την ακρίβεια μεταγραφής.

5.2 Μελλοντικές προεκτάσεις

Η συγκεκριμένη εφαρμογή είναι σε θέση για περαιτέρω ανάπτυξη τόσο σε επίπεδο λογισμικού όσο και λειτουργιών. Από πλευράς λογισμικού μπορεί να αναπτυχθεί και σε άλλες γλώσσες προγραμματισμού όπως η C++, όπου λόγω της αρχιτεκτονικής της θα είναι σε θέση να δίνει ταχύτερα αποτελέσματα αλλά και να τροποποιηθεί και ως plug-in για μπορεί να εισάγεται σε προγράμματα μουσικής παραγωγής και επεξεργασίας (DAW) .

Σε επίπεδο λειτουργιών θα μπορούσε να αναπτυχθεί ώστε το σύστημα να υλοποιεί πολυφωνική μεταγραφή για να μπορεί να μας παρέχει πληροφορία και σε περιπτώσεις που κατά την μουσική εκτέλεση συνηχούν νότες ή παίζονται συγχορδίες. Επιπλέον, θα μπορούσε

να δημιουργηθεί αλγόριθμος που να καταγράφει την όλη εκτέλεση και στο τέλος της να εξάγει ένα αρχείο ήχου που να την περιέχει για την μετέπειτα αξιοποίηση της. Επίσης, θα μπορούσαν να προστεθούν και άλλες τεχνικές σύνθεσης και εφέ.

Τέλος, αναμένεται ότι η εύρεση της τονικότητας θα μπορούσε να βελτιωθεί σημαντικά από τη χρήση τεχνικών μηχανικής μάθησης (machine learning), όπου θα απαιτείτο βέβαια και μία πιο λεπτομερή έρευνα όσον αφορά τα ακουστικά χαρακτηριστικά που εισάγονται στον αλγόριθμο αναγνώρισης. Από την παρούσα ανάλυση διαφαίνεται ότι η ενέργεια των peak φίλτρων είναι μια καλή επιλογή ως ακουστικό χαρακτηριστικό. Επίσης, σε αυτό θα μπορούσαν να προστεθούν και άλλα ακουστικά χαρακτηριστικά ώστε να βελτιωθεί η ακρίβεια στην μουσική μεταγραφή.

ΒΙΒΛΙΟΓΡΑΦΙΑ

Cook Perry (2002), *Real Sound Synthesis for Interactive Application*, Boca Raton, Florida: CRC Press.

Curtis Roads (1996), *The Computer Music Tutorial*, Cambridge, Massachusetts: The MIT Press.

Wes McKinney (2012), *Python for Data Analysis*. O'Reilly Media, Inc.

Eli Bressert (2012), *SciPy and NumPy: An Overview for Developers*, O'Reilly Media, Inc.

Werner Heisenberg (1930), *The Physical Principles Of The Quantum Theory*, Dover Publications, Inc.

Anders Vretblad (2003), *Fourier Analysis and Its Applications*. Springer.

Sophocles J. Orfanidis (1995), *Introduction to signal processing*, Prentice-Hall, Inc.

Xander Fiss (2011), *Real-Time Software Electric Guitar Audio Transcription*, Thesis. Rochester Institute of Technology

Αναστασία Γεωργάκη (2003), Σημειώσεις Μαθήματος Ανάλυση και Σύνθεση Ήχων. Τμήμα Μουσικών Σπουδών, Πανεπιστήμιο Αθηνών.

Praat - 'Digital Audio Workstation' – Διαθέσιμο στο: https://www.fon.hum.uva.nl/praat/

ΚΩΔΙΚΑΣ PYTHON

Το αρχείο κώδικα με τα συνοδευτικά αρχεία που χρειάζεται για να λειτουργήσει η εφαρμογή είναι για διαθέσιμα στον ιστότοπο του Git Hub.

https://github.com/ZacStl/Guitar-Conversion.git

```
1   import numpy as np
2   import pyaudio
3   import time
4   import threading
5   from scipy import signal
6   from scipy.io.wavfile import read
7   import tkinter as tk
8   from tkinter import ttk
9   from tkinter import filedialog
10  import os
11
12  class Stream:
13      def __init__(self, master):
14
15          self.master = master
16          self.rate = 44100
```

```
17          self.notes = [82.41, 87.31, 92.50, 98.00, 103.83, 110.00, 116.54, 123.47,
18                        130.00, 138.59, 146.83, 155.56, 164.81, 174.61, 185.00, 196.00]
19          self.Q = [9.0, 9.0, 9.0, 9.0, 9.0, 9.0, 9.0, 9.0,
20                    9.0, 9.0, 9.0, 9.0, 9.0, 9.0, 9.0, 9.0]
21          self.note_name = ['E2', 'F2', 'F#2/Gb2', 'G2', 'G#2/Ab2', 'A2', 'A#2/Bb2', 'B2',
22                            'C3', 'C#3/Db3', 'D3', 'D#3/Eb3', 'E3', 'F3', 'F#3/Gb3', 'G3']
23          self.Nnotes = len(self.notes)
24
25          self.Bs = np.zeros((self.Nnotes, 3))
26          self.As = np.zeros((self.Nnotes, 3))
27
28          self.IC = np.zeros((2, self.Nnotes))
29          self.rmsMatrix = np.array([])
30          self.rmsCurrentFrame = np.zeros((self.Nnotes,))
31
32          self.devIdxIN = 1
33          self.devIdxOUT = 3
34          self.CHANNELS = 1
35          self.CHUNK = 128
36          self.FORMAT = pyaudio.paFloat32
37
38          self.p = pyaudio.PyAudio()
39          self.len_devices = self.p.get_device_count()
40          self.len_devices_list = (range(self.len_devices))
41
42          self.volume = 0.8
43          self.guitar_vol = 0.2
```

```
44          self.wavfiles = ['BassE1', 'TromboneE2', 'Man_voice', 'Select file']
45
46          self.pitchnote = ['E1', 'F1', 'F#1/Gb1', 'G1', 'G#1/Ab1', 'A1', 'A#1/Bb1', 'B1',
47                            'C2', 'C#2/Db2', 'D2', 'D#2/Eb2', 'E2', 'F2', 'F#2/Gb2', 'G2']
48
49          self.currentIndex = 0.0
50          self.currentSample = 0.0
51          self.amp = 0.0
52
53          self.alpha = 0.002
54          self.output = np.zeros((self.CHUNK,))
55
56          self.paused = True
57          self.playing = False
58
59
60
61          self.iir_peak()
62          self.buttons()
63          self.change_wavetable()
64
65
66
67      def buttons(self):
68          '''frame0'''
69          frame0 = tk.Frame(self.master, bg='#659d79')
70          self.var1 = tk.StringVar(self.master)
71          self.label_note = tk.Label(frame0, textvariable=self.var1, width=38, height=6,
72                                     font=('calibri', 20)).pack(padx=40, pady=40)
```

```
73          self.button_2 = tk.Button(frame0, text='stop', width=13, height=1,
74                                    relief='solid', command=self.pause).pack(side='bottom')
75
76
77          self.button = tk.Button(frame0, text='start', width=13, height=1,
78                                    relief='solid', command=self.play).pack(side='bottom')
79
80
81          frame0.pack(side='top', fill='both')
82
83        '''frame2'''
84
85          frame2 = tk.Frame(self.master, bg='#659d79')
86
87          self.Guitar_vol = tk.Scale(frame2, from_=6, to_=-80,
88                                     label="Guitar Volume", length=200, resolution=1.0,
89                                     command=self.set_guitar_vol, highlightcolor="red", fg='red',
90                                     font=('calibri', 8), bg="#838B8B", orient="vertical")
91          self.Guitar_vol.set(self.guitar_vol)
92          self.Guitar_vol.place(relx=0.18, rely=0.5, anchor='center')
93
94
95          self.Alpha = tk.Scale(frame2, from_=-20, to_=-80,
96                                     label="Alpha", length=200, resolution=1.0,
97                                     command=self.set_alpha, highlightcolor="red", fg='red',
98                                     font=('calibri', 8), bg="#838B8B", orient="vertical")
99          self.Alpha.set(self.alpha)
100         self.Alpha.place(relx=0.43333, rely=0.5, anchor='center')
```

```
101         self.Volume = tk.Scale(frame2, from_=6, to_=-80,
102                                label="Volume", length=200, resolution=1.0,
103                                command=self.set_volume, highlightcolor="red", fg='red',
104                                font=('calibri', 8), bg="#838B8B", orient="vertical")
105         self.Volume.set(self.volume)
106         self.Volume.place(relx=0.66, rely=0.5, anchor='center')
107 
108         frame2.pack(side='right', expand=True, fill='both')
109 
110       '''frame1'''
111 
112         frame1 = tk.Frame(self.master, bg='#659d79')
113 
114 
115         self.var0 = tk.StringVar(root)
116         self.Pitch_height = tk.OptionMenu(
117             frame1, self.var0, *self.pitchnote).place(relx=0.57, rely=0.23, anchor='center')
118         self.var0.trace('w', self.set_pitch_height)
119 
120 
121         self.var = tk.StringVar(root)
122         self.var.set(self.wavfiles[0])
123         self.dropmenu = tk.OptionMenu(
124         frame1, self.var, *self.wavfiles).place(relx=0.38, rely=0.23, anchor='center')
125         self.var.trace('w', self.change_wavetable)
```

```
126         self.var3 = tk.StringVar(root)
127         self.var3.set(self.devIdxIN)
128         self.choose_inport = tk.OptionMenu(
129             frame1, self.var3, *self.len_devices_list).place(relx=0.358, rely=0.7, anchor='center')
130         self.var3.trace('w', self.in_port)
131
132
133         self.var4 = tk.StringVar(root)
134         self.var4.set(self.devIdxOUT)
135         self.choose_output = tk.OptionMenu(
136             frame1, self.var4, *self.len_devices_list).place(relx=0.56, rely=0.7, anchor='center')
137         self.var4.trace('w', self.out_port)
138
139
140         self.devices = tk.Button(frame1,  text='See Available Ports', command=self.choose_ports,
141                                  relief='solid',
142                                  font=('calibri', 12)).place(relx=0.457, rely=0.48, anchor='center')
143
144
145         frame1.pack(side='right', expand=True, fill='both')
146
147
148     def choose_ports(self):
149         top = tk.Toplevel()
150         top.geometry("850x600")
151         top.configure(background='black')
152         top.title("In-Out Ports")
```

```
153         top.grid_columnconfigure(0, weight=1)
154         top.grid_rowconfigure(0, weight=1)
155
156         text = tk.Text(top, height=60, width=200,
157                        pady=10, padx=10, bg='orange')
158         text.grid(row=0, column=0, sticky=tk.NW)
159
160         scrollbar = ttk.Scrollbar(top, orient='vertical', command=text.yview)
161         scrollbar.grid(row=0, column=1, sticky=tk.NS)
162
163         text['yscrollcommand'] = scrollbar.set
164
165         gap = 0
166         for dev in range(self.p.get_device_count()):
167
168             nam = self.p.get_device_info_by_index(dev).get('name')
169             inf = self.p.get_device_info_by_index(dev)
170             dev_num = 'device=' + str(dev)
171             posision = f'{gap}.0'
172             printed = f'{nam}\n{inf}\n{dev_num}\n\n'
173
174             text.insert(posision, printed)
175             gap += 5
176
177         top.mainloop()
```

```
178     def set_volume(self, value):
179         self.volume = float(10**(float(value)/20))
180         return self.volume
181
182
183     def set_guitar_vol(self, value):
184         self.guitar_vol = float(10**(float(value)/20))
185         return self.guitar_vol
186
187
188
189     def set_alpha(self, value):
190         self.alpha = float(10**(float(value)/20))
191         return self.alpha
192
193
194
195     def in_port(self, *args):
196         current_inport = self.var3.get()
197         return int(current_inport)
198
199
200
201     def out_port(self, *args):
202         current_outport = self.var4.get()
203         return int(current_outport)
```

```
204     def set_pitch_height(self, *args):
205         current0 = self.var0.get()
206 
207         if current0 == 'E1':
208             self.pitch = 41.2
209 
210         elif current0 == 'F1':
211             self.pitch = 43.65
212 
213         elif current0 == 'F#1/Gb1':
214             self.pitch = 46.25
215 
216         elif current0 == 'G1':
217             self.pitch = 49.00
218 
219         elif current0 == 'G#1/Ab1':
220             self.pitch = 51.91
221 
222         elif current0 == 'A1':
223             self.pitch = 55.00
224 
225         elif current0 == 'A#1/Bb1':
226             self.pitch = 58.27
227 
228         elif current0 == 'B1':
229             self.pitch = 61.74
230 
231         elif current0 == 'C2':
232             self.pitch = 65.41
```

```
233         elif current0 == 'C#2/Db2':
234             self.pitch = 69.30
235
236         elif current0 == 'D2':
237             self.pitch = 73.42
238
239         elif current0 == 'D#2/Eb2':
240             self.pitch = 77.78
241
242         elif current0 == 'E2':
243             self.pitch = 82.41
244
245         elif current0 == 'F2':
246             self.pitch = 87.31
247
248         elif current0 == 'F#2/Gb2':
249             self.pitch = 92.50
250
251         elif current0 == 'G2':
252             self.pitch = 98.00
253
254         return self.pitch
255
256     '''Definition of IIR peak filters'''
257     def iir_peak(self):
258         for idx, freq in enumerate(self.notes):
259             B, A = signal.iirpeak(freq, self.Q[idx], int(self.rate))
260             self.Bs[idx] = B
261             self.As[idx] = A
```

```
262     ''' Wavetables '''
263     def change_wavetable(self, *args):
264
265         curent = self.var.get()
266
267         if curent == 'BassE1':
268             self.currentIndex = 0.0
269             self.currentSample = 0.0
270             self.var0.set(self.pitchnote[0])
271             self.samplerate, self.wave_table = read('E1662_44.wav')
272             self.waveTable = self.wave_table / 32000.0
273             self.waveTable_length = len(self.waveTable)-1
274
275         elif curent == 'TromboneE2':
276             self.currentIndex = 0.0
277             self.currentSample = 0.0
278             self.var0.set(self.pitchnote[6])
279             self.samplerate, self.wave_table = read('tromboneE2.wav')
280             self.waveTable = self.wave_table / 32000.0
281             self.waveTable_length = len(self.waveTable)-1
282
283         elif curent == 'Man_voice':
284             self.currentIndex = 0.0
285             self.currentSample = 0.0
286             self.var0.set(self.pitchnote[12])
287             self.samplerate, self.wave_table = read('man_voice_Bb.wav')
288             self.waveTable = self.wave_table / 32000.0
289             self.waveTable_length = len(self.waveTable)-1
```

```
290         elif curent == 'Select file':
291             try:
292                 file_select = filedialog.askopenfilename(initialdir='/',
293                                         title='Select file',
294                                         filetypes=(('wav files', '*.wav'), ('All files', '*.*')))
295                 basename = os.path.basename(file_select)
296                 self.currentIndex = 0.0
297                 self.currentSample = 0.0
298                 self.var0.set(self.pitchnote[0])
299                 self.samplerate, self.wave_table = read(str(basename))
300                 self.waveTable = self.wave_table / 32000.0
301                 self.waveTable_length = len(self.waveTable)-1
302             except:
303                 ValueError
304 
305     '''Interpolation'''
306     def interpolate_linearly(self, inote):
307         tableDelta = inote / float(self.set_pitch_height())
308         index0 = int(self.currentIndex)
309         index1 = index0 + 1
310         frac = self.currentIndex - index0
311         value0 = self.waveTable[index0]
312         value1 = self.waveTable[index1]
313         self.currentSample = value0 + frac * (value1-value0)
314         self.currentIndex += tableDelta
315         if self.currentIndex > self.waveTable_length:
316             self.currentIndex -= self.waveTable_length
317 
318         return self.currentSample
```

```
319    '''Pyaudio Callback Function'''
320    def callback(self, in_data, frame_count, time_info, status):
321
322        sIN = np.frombuffer(in_data, dtype=np.float32)
323
324        for i in range(self.Nnotes):
325            filtsig, ic = signal.lfilter(
326                    self.Bs[i], self.As[i], sIN, zi=self.IC[:, i])
327            self.IC[:, i] = ic
328            self.rmsCurrentFrame[i] = np.linalg.norm(filtsig)
329
330        maxpos = np.argmax(self.rmsCurrentFrame)
331        printed = self.note_name[maxpos]
332        self.var1.set(printed)
333        inote = self.notes[maxpos]
334        inote *= 0.5
335
336        for n in range(self.CHUNK):
337            self.amp = self.amp*(1-self.alpha) + self.alpha*np.abs(sIN[n])
338            self.output[n] = self.interpolate_linearly(inote) * self.amp
339
340
341        self.output = self.volume * (self.output + self.guitar_vol *sIN)
342
343        return (self.output.astype(np.float32), pyaudio.paContinue)
```

```
344     ''' Setup Pyaudio '''
345     def Pyaudio(self):
346         p = pyaudio.PyAudio()
347         stream = self.p.open(format=self.FORMAT,
348                              channels=self.CHANNELS, input_device_index=self.in_port(),
349                              output_device_index=self.out_port(),
350                              rate=self.rate,
351                              input=True,
352                              output=True,
353                              frames_per_buffer=self.CHUNK,
354                              stream_callback=self.callback)
355
356         stream.start_stream()
357         while stream.is_active and self.playing:
358             if self.pause:
359                 time.sleep(0.001)
360             else:
361                 time.sleep(0.001)
362
363         self.playing = False
364         stream.close()
365         p.terminate()
366
367
368     def pause(self):
369         p = pyaudio.PyAudio()
370         self.paused = True
371         self.playing = False
372         p.terminate()
```

```
373     def pause(self):
374         p = pyaudio.PyAudio()
375         self.paused = True
376         self.playing = False
377         p.terminate()
378 
379     def play(self):
380         if not self.playing:
381             self.playing = True
382             threading.Thread(target=self.Pyaudio, daemon=True).start()
383             self.paused = False
384         self.paused = False
385 
386     def stop(self):
387         self.playing = False
388 
389 def handle_close():
390     player.stop()
391     root.destroy()
392 
393 ''' SETUP AND RUN '''
394 root = tk.Tk()
395 root.title("Guitar Conversion")
396 root.configure(background='black')
397 root.geometry("800x600")
398 root.iconbitmap('logowhite.ico')
399 player = Stream(root)
400 root.protocol("WM_DELETE_WINDOW", handle_close)
401 root.mainloop()
```