POLITECNICO DI TORINO

Corso di Laurea Magistrale in
Ingegneria del Cinema e dei Mezzi di Comunicazione

Tesi di Laurea Magistrale

Sviluppo di uno strumento software per la rappresentazione grafica in tempo reale delle sensazioni sonore espresse da brani musicali

Relatore

prof. Antonio Servetti

Candidato

Francesco Cretti

Luglio 2017

Ringraziamenti

A Valentina Maistri, che forse dovrebbe scrivere qui lei i propri ringraziamenti, perché il suo aiuto è stato semplicemente imprescindibile, ed è come se questo lavoro fosse stato scritto a quattro mani.

Al mio relatore, Prof. Antonio Servetti, che con pazienza e disponibilità ha messo luce sulle cose quando queste si facevano confuse.

Ad Ivana ed Oreste, che non hanno mai mancato di supportarmi completamente nel mio percorso, ma che sopratutto mi hanno privilegiato di un rapporto di rispetto e stima reciproca.

A Giovanni Bologni, che si è rivelato, oltre che un competente compagno di lavoro, un ottimo amico ed una persona con cui condivido molti aspetti della vita.

A Nicola Ruffino, Luciano Prono, Andrea Gambedotti e Lorenzo Luca, per aver creduto nel progetto Dreamo e per aver impiegato risorse, tempo ed energie nel perseguire le nostre idee.

Ad Andrea Fiscella, che ha visto in Dreamo un progetto interessante e che con serietà sta fornendo il suo importante contributo.

A Quique Valido e Michele Cattani, i migliori compagni di studio, di lavoro e di avventure che avrei potuto sperare.

A Paolo Urbani, Paolo Pilati e Giuliano Buratti, per le centinaia (ma anche migliaia) di chilometri passati ad andare in lungo e in largo, e per portare avanti con dedizione un progetto che spesso mi ha dato forza e motivazione per tenere la testa alta e superare le difficoltà.

Ai colleghi Francesco Campo, Kenta Crisà, Renato Angelone, Valentina Valle e Mateusz Polek, a cui auguro nel futuro di continuare a coltivare i loro talenti e le loro capacità.

A Marco Ziccardi, per una chiacchierata davanti ad un paio di birre che si è rivelata più utile di quanto mi aspettassi.

A Francesco Francioso e Stefano Viotto, che hanno avuto il delicato compito di accogliermi in questa città e lo hanno fatto nel migliore dei modi.

A Roberto, Marco Sirio, Benny ed a tutti i miei amici per ogni momento passato a discutere, analizzare, discorrere, fantasticare o bighellonare. Per ogni momento, serio o frivolo, passato sinceramente insieme che ha poi composto l'interezza della mia vita.

Sommario

1 Introduzione.....1
1.1 Nascita del progetto.....1
1.2 Il Team di Lavoro.....1
1.3 La progettualità studentesca.....1
1.4 Finalità ed obiettivi di questa tesi.....2
2 Il progetto Dreamo 5
2.1 Premessa: l'arte generativa ed il mondo dei makers.....5
2.2 Cos'è Dreamo 6
2.3 FabLab.....7
2.4 Struttura hardware 8
2.5 Struttura software.....9
2.5.1 Struttura del software di generazione video.....9
2.6 Altri strumenti.....10
3 Stato dell'arte.....11
3.1 Music information retrieval.....11
3.1.2 Music Mood Recognition.....14
3.1.3 I modelli dell'emozione: parametrico vs. categorico.....15
3.2 La visualizzazione del suono: esempi e progetti artistici 16
3.3 Strumenti e tools per ambienti audio/visuali.....21
3.3.1 Software ed ambienti di sviluppo 21
3.3.2 Sintetizzatori video analogici 22
4 L'analisi dell'audio.....25
4.1 Un'applicazione in real time.....25
4.2 Il motore di analisi.....26
4.3 La scelta delle features.....27
4.3.1 Features dinamiche.....27
4.3.2 Features timbriche.....28
4.3.3 Features ritmiche.....31
4.4 Lo "stato" della traccia 35
4.5 L'interfaccia utente.....37
5 Interazioni audio/video.....39

5.1 Interazioni mood oriented 39
5.2 Realizzazione di un video responsive 42
5.2.1 Assegnazioni dirette 42
5.2.2 Utilizzo dello stato della traccia 46
5.3 Creazione di una grafica di esempio 48
5.3.1 Forma di base 48
5.3.2 Gestione del colore 49
5.3.3 Gestione dei movimenti 55
5.3.4 Gestione delle linee 56
5.3.5 Gestione della forma/distorsione 60
5.3.6 Esempio di funzionamento 61
5.4 Dreamo come strumento: tutorial di utilizzo 67
5.4.1 Passaggio 1: creazione di una particella 67
5.4.2 Passaggio 2: creazione di una scena 68
5.4.3 Passaggio 3: impostazione dei meccanismi di interazione 70
6 Conclusioni 79
6.1 Dreamo ad oggi 79
6.2 Sviluppi futuri 80

7 Bibliografia 81
Appendice A 85
Appendice B 108
Appendice C 117
Appendice D 123

Capitolo 1

Introduzione

1.1 Nascita del progetto

Il progetto “Dreamo” nasce a metà del 2016, da un’idea di tre studenti del Politecnico di Torino: Nicola Ruffino (Ingegneria Biomedica), Giovanni Bologni (Ingegneria Elettronica) ed il sottoscritto, e si sviluppa attorno all’idea di creare un sistema che sia in grado di capire lo stato d’animo di una performance musicale e di tramutarlo in un video.
Partendo dal concetto di “arte generativa”, si è ragionato nell’ottica di congiungere le capacità tecniche ed ingegneristiche alle inclinazioni artistiche che muovono ogni componente del nostro team.
La passione condivisa per la musica è stata la molla di questo progetto, che è diventato un’occasione per sfruttare e sviluppare competenze sia dal punto di vista ingegneristico che musicale.

1.2 Il Team di Lavoro

Quando l’idea del Dreamo è nata è apparso chiaro che per la sua realizzazione sarebbe stato necessario trovare altre persone disposte a spendere tempo ed energie in questa scommessa. Per questo motivo il team di lavoro si è presto arricchito di un nuovo componente: Luciano Prono, anch’egli studente del Politecnico, in particolare del corso Magistrale di Ingegneria Elettronica.

1.3 La progettualità studentesca

I primi mesi di lavoro sono stati dedicati all’analisi e allo studio della letteratura, in

questa fase le specifiche del progetto sono state oggetto di innumerevoli modifiche, ed hanno portato alla teorizzazione del primo prototipo.
A settembre 2016 il progetto Dreamo èstato presentato al bando della Progettualità Studentesca ottenendo un finanziamento destinato alla realizzazione appunto di tale prototipo.

1.4 Finalità ed obiettivi di questa tesi

Lo scopo del progetto Dreamo è leggere lo "stato d'animo" della performance e generare degli effetti grafici coerenti con esso: gli aspetti legati alla natura del progetto nella sua totalità saranno trattati nel capitolo 2.

Questa sezione si concentra sugli obiettivi specifici della tesi: data la complessità del progetto, lo svolgimento di questo lavoro si è rivelato un'ottima occasione per definire e realizzare una prima versione del sistema e mettere così delle solide fondamenta per i passaggi successivi.
Questi obiettivi, per sintetizzare, sono:

- lo sviluppo di un motore di analisi per l'estrazione di features acustiche dallo stream audio;

- la progettazione di un meccanismo d'interazione fra features audio e parametri visivi;

- la generazione di una grafica esempio *real time* coerente con le caratteristiche dell'input sonoro, e reattiva ai cambiamenti dello stesso;

Il primo di questi punti viene trattato nel capitolo 4, mentre il secondo ed il terzo vengono esposti nel capitolo 5. Questo percorso ha richiesto la trattazione anche di aspetti diversi dall'elaborazione di segnali audio, come ad esempio ambiti legati al graphic design, alla psicologia e parzialmente della gestione della sensoristica, rivelandosi così un'ottima occasione di formazione personale.

La revisione della letteratura, descritta nel capitolo 3, ha consentito di comprendere

le numerose tecniche già presenti in questo ambito, ed avere coscienza di ciò che si potesse adattare meglio alla natura del progetto Dreamo. A partire da queste premesse è stato impostato e sviluppato il lavoro.
Questa tesi rappresenta quindi il raggiungimento di un primo step e si pone come punto di partenza per sviluppi futuri, a partire dall'integrazione dei sensori biometrici, che attualmente non vengono utilizzati per influenzare il comportamento grafico, e dalla creazione di un modello di mood più completo.

Dreamo segue l'idea di essere uno strumento utilizzabile da vari artisti che, a seconda del grado di competenza, possono addentrarsi nella personalizzazione del sistema per produrre risultati unici e particolari.

Anche il lavoro di questa tesi si pone come uno strumento aperto che abbraccia l'intervento di chiunque abbia voglia non solo di sfruttare, ma anche di aumentare, le potenzialità di questo progetto. A conferma di questo, nell'ultima sezione del capitolo 5 è disponibile un breve tutorial su come creare una grafica animata dall'audio, sfruttando le funzionalità implementate fino a questo punto.

Capitolo 2

Il progetto Dreamo

In questo capitolo verrà esposto il progetto Dreamo nella sua interezza, con i suoi obiettivi finali, nel contesto del quale si colloca il lavoro svolto per questa tesi, come descritto nel paragrafo 1.4.

2.1 Premessa: l'arte generativa ed il mondo dei makers

Philip Galanter ha scritto:

"*Per arte generativa si intende ogni pratica artistica in cui l'artista usi un sistema, come un programma informatico, una macchina o qualsiasi altra invenzione procedurale, attivato secondo un certo grado di autonomia che contribuisce a - o produce - un'opera d'arte finita*".

Attorno a questo concetto nel corso degli ultimi anni sono nati numerosi progetti, sostenuti e supportati anche dalla rete dei makers, i c.d. "Artigiani digitali". Attraverso questa comunità internazionale infatti, informazioni e conoscenze vengono veicolate e trasmesse sia sfruttando il web che avvalendosi di veri e propri luoghi fisici (ad esempio i Fab Lab [1]).

La filosofia dei makers è inscindibile dal protocollo open-source, che favorisce una diffusione del sapere rapida e trasparente, sfruttando portali quali GitHub e Instructables; in questo modo le idee innovative arrivano in ogni angolo connesso del globo, favorendo la crescita dei singoli progetti e lo scambio di opinioni.

Inoltre il campo dell'interattività è di sempre maggior rilievo e costituisce un trend importante nella ricerca scientifica più recente.

2.2 Cos'è Dreamo

Facendo di tali premesse la nostra filosofia di lavoro, abbiamo sviluppato l'idea di un generatore di video real-time, basato su sensoristica per biosegnali e su ingressi audio.
Il nostro obiettivo è quindi quello di realizzare un dispositivo stand-alone per la generazione semiautomatica di video astratti, utilizzabili in performance musicali ed artistiche, che conferiscano forza e valore all'esibizione.
Per questo abbiamo pensato ad un sistema composto da un bracciale indossabile dal musicista/performer, che comunichi con un microcontrollore centrale (ad esempio un Raspberry Pi) equipaggiato con uscita video da collegare ad un monitor o videoproiettore.

L'obiettivo primario del progetto è quindi la generazione di un risultato visivo che scaturisca dall'informazione raccolta dalla sensoristica e che sia coerente con lo stato d'animo della performance.

A partire dai dati ricavati il sistema il sistema dovrà quantificare lo stato d'animo (sia quello espresso della musica che quello del performer) e sulla basi di principi estetici e psicologici, generare un video astratto, modificandone in tempo reale parametri grafici. Gli obiettivi scientifici sono, oltre a creare degli effetti visuali artisticamente significativi, lo studio di un algoritmo di riconoscimento dello "stato emotivo" dell'utente e della musica, che riesca ad operare scelte in modo intelligente.
Dreamo vuole essere quindi uno strumento semi automatico a disposizione di qualunque artista o performer che voglia cimentarsi nella sperimentazione di interazioni audiovideo, e voglia fare affidamento su un sistema che è già in grado di quantificare le emozioni di una performance musicale.
Dreamo può rappresentare un approccio differente nel vasto mondo degli ambienti audio/visuali.

Fig 2.1 Logo di Dreamo

2.3 FabLab

Il Dreamo ha trovato la sua localizzazione geografica nelle officine del FabLab[1] di Torino.

Il FabLab (Fabrication Laboratory) è un laboratorio di fabbricazione digitale che nasce con l'intento di sostenere una rete globale in cui condividere strumenti e processi creativi il quale ospita spesso workshop e corsi, ma anche piccoli gruppi di lavoro come il nostro.

Il concetto di laboratorio condiviso ha permesso a molti progetti di trovare un supporto, sia logistico che tecnologico, fondamentale per le prime fasi di sviluppo. Inoltre, un ambiente florido e stimolante come quello del FabLab ha favorito il confronto con persone che coltivano l'interesse comune per la tecnologia, le quali hanno messo in luci aspetti e punti di vista diversi del nostro progetto.

Una di queste persone è sicuramente Andrea Gambedotti, che ci ha fornito un supporto importante grazie alla sua esperienza nel campo della comunicazione grafica e del web design. È infatti di sua creazione il logo del progetto Dreamo.

1 Maggiori informazioni su http://fablabtorino.org/

2.4 Struttura hardware

La struttura hardware risulta abbastanza semplice, ed è strutturata nel seguente modo: i sensori per biosegnali vengono gestiti dalla shield MySignals HW[2] , applicata ad un microcontrollore Arduino Uno. Il software di generazione video è attualmente installato su laptop, in particolare su un MacBook Pro equipaggiato con Intel Core i5, l'ingresso audio analogico è gestito dalla scheda audio integrata nel computer. Allo stesso modo l'uscita video è quella della scheda grafica del computer.
Negli sviluppi futuri è prevista l'installazione del programma su scheda Raspberry Pi3 e la configurazione di un collegamento wireless (Bluetooth o WiFi direct) con Arduino Uno.

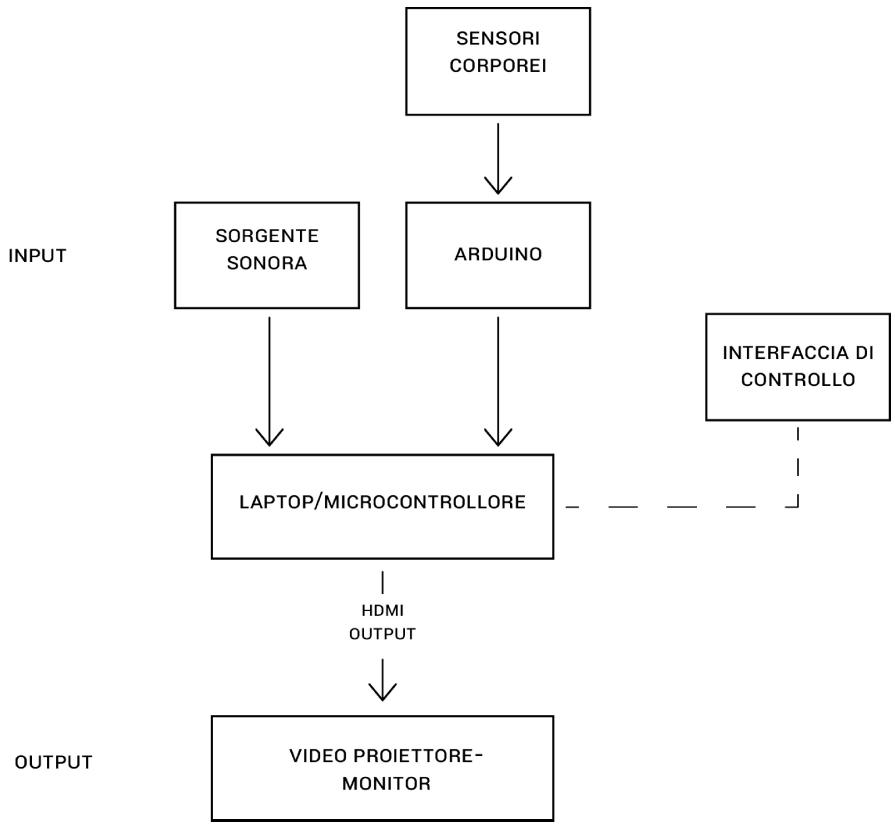


Fig. 2.2 Schema a blocchi della struttura hardware

2 Maggiori informazioni su http://www.my-signals.com/

2.5 Struttura software

Date nel necessità e le caratteristiche del progetto, la scelta della piattaforma software sulla quale appoggiarsi è ricaduta su Processing.
Processing è un framework open source ed un linguaggio di programmazione orientato allo sviluppo grafico, creato da Ben Fry e Casey Reas [2], in costante crescita e sviluppo. Questo linguaggio è adatto alla creazione di prodotti grafici interattivi, e contiene diverse librerie con funzioni di disegno di forme grafiche di base (quadrati, cerchi, linee, testi…), di gestione del colore, e di altre funzionalità importanti per la creazione di effetti visuali.

Dal punto di vista dell'audio processing, sono disponibili sul web diverse librerie di terze parti disponibili per la piattaforma Processing, ma nessuna di esse può dirsi esaustiva dal punto di vista dell'estrazione di descrittori dall'audio. La più completa è sicuramente la libreria Minim[3], che fornisce funzioni per gestire e bufferizzare lo stream audio ed effettuare operazioni di analisi audio quali l'FFT.
Questa libreria è stata utilizzata come base di partenza per sviluppare poi tutte le funzionalità necessarie per il funzionamento del nostro progetto. Il motore di analisi audio sarà esposto dettagliatamente nel capitolo 4.

2.5.1 Struttura del software di generazione video

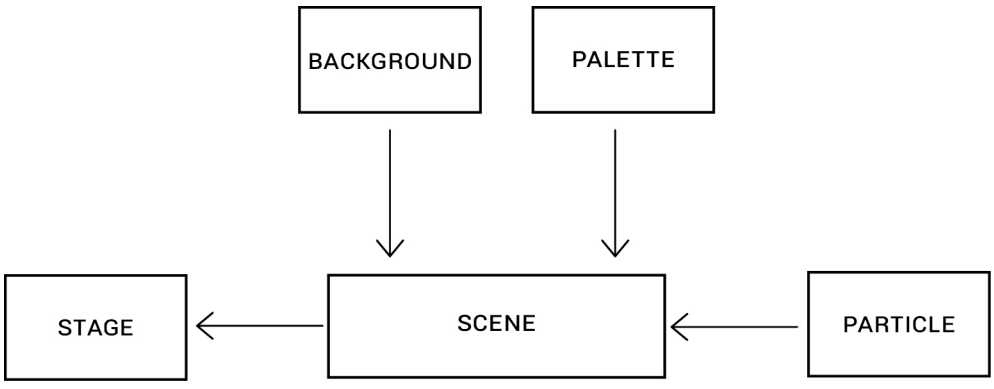


Fig.2.3 Schema a blocchi della struttura software del motore di generazione video

3 Maggiori informazioni su https://github.com/ddf/Minim

La struttura di generazione degli effetti visuali è stata pensata per essere versatile e per permettere la creazione di effetti molto diversi. In seguito elencate le classi principali che compongono il software di rendering.

- **Particle** - la classe particella rappresenta il nucleo del sistema visuale. Contiene i metodi di tracciamento ed aggiornamento degli elementi grafici di base. Dalla classe Particle possono essere ereditate numerose sottoclassi con caratteristiche e regole differenti.

- **Scene** - la classe scena include un array di particelle e gestisce la regole con cui esse vengono tracciate ed aggiornate. Dalla classe Scene generale possono essere ereditate numerose sottoclassi con regole differenti, riguardanti il numero o il tipo di particelle. Gestisce anche lo sfondo dell'effetto grafico e si occupa di passare la palette di colori corrente a tutte le particelle che contiene.

- **Stage** - questa classe include un array di scene, e contiene i metodi che permettono di passare da una scena all'altra

Questa struttura permette di generare ed aggiornare scenari grafici, implementare effetti, gestire i passaggi e le transizioni, nonché interagire in modo flessibile con gli input esterni, ottenuti dalle classi dedicate all'analisi dell'audio.
Il codice dell'intero programma è disponibile all'indirizzo: github.com/Dreamo-Italy/Dreamo-engine-ProcessingIDE/tree/feature/thesisPresentation

2.6 Altri strumenti

Durante la fase di sviluppo degli algoritmi è stato sfruttato l'ambiente di calcolo MATLAB per l'analisi statistica dei dati ed il plottaggio di grafici, mentre per l'esecuzione dei test degli algoritmi scritti in Java è stato utilizzato l'ambiente di sviluppo Eclipse.

Capitolo 3

Stato dell'arte

In questo capitolo verranno analizzati gli studi disponibili in letteratura e i progetti correlati all'ambito di interesse di questa tesi.
In particolare: nel paragrafo 3.1 sono presentate le tecniche di estrazione di features acustiche da tracce audio, nonché gli studi eseguiti in merito alla rappresentazione dell'emozione attraverso modelli matematici e le tecniche di machine learning associate.
Il secondo paragrafo è, invece, dedicato a quei progetti artistici che hanno esplorato il mondo della rappresentazione visuale della musica, tenendo conto dei diversi approcci sia artistici che tecnici.
Il terzo ed ultimo paragrafo si concentra sugli strumenti orientati alla generazione di effetti audio/visuali, già presenti in commercio.

3.1 Music information retrieval

Le tecniche di analisi dei contenuti audio fanno parte dell'ambito detto *music information retrieval*, ovvero l'insieme dei metodi che consentono di estrarre informazioni di diversa natura da dati audio.

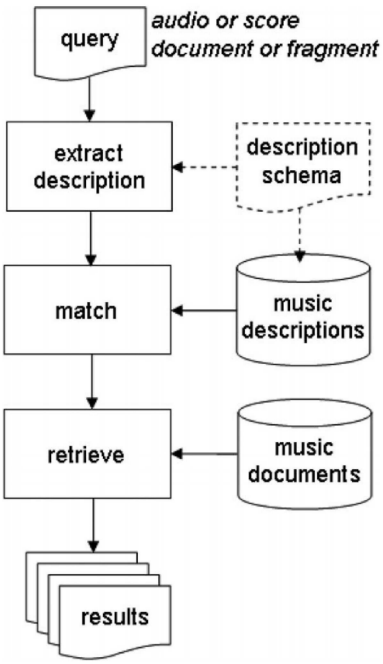


Fig. 3.1: Workflow di un generale sistema di MIR tratto da [3]

Questo campo di ricerca sta suscitando interesse crescente negli ultimi tempi, e trova un crescente numero di applicazioni, [3] come ad esempio l'organizzazione di database musicali basati su caratteristiche acustiche [4,5], la classificazione per genere musicale, per mood (aspetto che sarà approfondito in seguito), l'identificazione dell'artista o dello strumento. [6]

Use Case	Specificity	Description
Music Identification	H	Identify a compact disk, provide metadata about an unknown track, mobile music information retrieval: e.g. *shazam.com*
Plagiarism detection	H	Identify mis-attribution of musical performances, mis-appropriation of music intellectual property.
Copyright monitoring	H	Monitor music broadcast for copyright infringement or royalty collection
Versions	H/M	Remixes, live vs. studio recordings, cover songs. Used for database normalization and near-duplicate results elimination
Melody	H/M	Find works containing a melodic fragment
Identical Work / Title	M	Retrieve performances of same opus number or song title
Performer	M	Find music by a specific artist
Sounds like	M	Find music that sounds like a given recording
Performance Alignment	M	Mapping one performance onto another independent of tempo and repetition structure
Composer	M	Find works by one composer
Recommend-ation	M/L	Find music that matches the user's personal profile
Mood	L	Find music using emotional concepts: *Joy, Energetic, Melancholy, Relaxing*
Style / Genre	L	Find music that belongs to a generic category: *Jazz, Funk, Female Vocal*
Instrument(s)	L	Find works with same instrumentation
Music-Speech	L	Radio broadcast segmentation, Music archives cataloguing

Fig. 3.2: Esempi di task MIR d'interesse tratto da [3]

A tal proposito viene organizzato annualmente il MIREX (music information research evaluation exchange), un meeting che si concentra su algoritmi e metodi di *music information retrieval. [7]*

3.1.1 Estrazione di features acustiche

I task di *music information retrieval* sono stati affrontati con diversi approcci [3], uno dei quali è l'estrazione di *features* acustiche di basso livello. È possibile studiare l'andamento del timbro, del ritmo, della dinamica, della melodia e di altre caratteristiche

di un frammento musicale, estraendo informazioni a livello di frames (solitamente di lunghezze comprese fra i 10 ed i 1000 ms) o analizzandone le loro distribuzioni statistiche.

Al giorno d'oggi, numerosi studi sull'analisi dell'audio sono disponibili in letteratura, e diversi strumenti software sono distribuiti in versione commerciale o *open-source*. Testi importanti come quello di Lerch [8] o [9] sono stati un buon punto di partenza per il lavoro presentato con questa tesi. Un altro ottimo riferimento è stato individuato del lavoro di Tzanetakis, il quale ha esplorato i diversi aspetti e tecniche nell'ambito del MIR, creando il framework Marsyais in un secondo momento [10].

Esistono inoltre diversi algoritmi di analisi acustica proposti in letteratura spesso riuniti in pacchetti software scritti in diversi linguaggi come C++ [11], Matlab [12] o Java [13].

3.1.2 Music Mood Recognition

Quantificare l'emozione (non solo per quanto riguarda la musica) è un *task* difficile e stimolante per molti ricercatori e scienziati. È innanzitutto necessaria una teoria psicologica alla base, che permetta la scelta di un modello emozionale accurato e coerente.

Appare necessario tenere sempre presente come le emozioni siano di difficile interpretazione, le quali dipendono non solo dal contesto culturale e sociale ma variano anche da individuo ad individuo: ciò significa che anche in contesti assimilabili è la soggettività di ognuno a emergere. I modelli proposti sono quindi un tentativo di approssimare il calcolo delle emozioni, ognuno concentrandosi sul proprio obiettivo specifico e tralasciando inevitabilmente altre variabili in gioco.

La *music emotion*[4] *recognition* (MER), costituisce un ambito di ricerca ancora primitivo, nel quale non vi è ancora una teoria unificata ed è stato incluso nei task del MIREX solo a partire dal 2007 [14].

Una delle difficoltà consistono nell'ottenere una *ground truth* affidabile, a causa della già citata soggettività e variabilità delle impressioni emotive riguardo alla musica, nonché sulla difficoltà nel scegliere un metodo di annotazione adeguato [6,15]. Sicuramente, la

4 Emotion e mood sono termini equivalenti in questo contesto

ricerca psicologica alla base del MER propone un'importante distinzione fra le emozioni espresse e quelle indotte dalla musica [16]: l'oggetto di questi studi, si concentra sull'emozione *espressa* dalla musica, piuttosto che su quella *indotta*, che risulterebbe troppo complessa da quantificare.

Inoltre studi sul significato culturale della musica suggeriscono che potrebbero esserci reazioni alla musica comuni attraverso le varie culture popolazioni e suggeriscono quindi che sia sensata la ricerca di un sistema di quantificazione dell'emozione nella musica [17,18].

3.1.3 I modelli dell'emozione: parametrico vs. categorico

Esistono due principali proposte di modelli matematici per descrivere le emozioni: un modello categorico ed un modello continuo.
Il modello categorico punta a seprare le emozioni in cluster che contengono aggettivi strettamente correlati fra loro. Il music emotion recognition diventa in questo caso un problema di classificazione, e viene affrontato con tecniche che includono l'utilizzo di strumenti quali i Gaussian Mixtures Models (GMMs) [19] o di metodologie quali le Macchine a vettori di supporto (*Support Vector Machines* o SVM) [20].

Clusters	Mood Adjectives
Cluster 1	passionate, rousing, confident, boisterous, rowdy
Cluster 2	rollicking, cheerful, fun, sweet, amiable/good natured
Cluster 3	literate, poignant, wistful, bittersweet, autumnal, brooding
Cluster 4	humorous, silly, campy, quirky, whimsical, witty, wry
Cluster 5	aggressive, fiery, tense/anxious, intense, volatile, visceral

Fig. 3.3: Aggettivi utilizzati per il task di classificazione del mood del MIREX [17]

Il secondo approccio invece suggerisce che il mood può essere rappresentato e misurato da un descrittore continuo, mappato su un piano multi-dimensionale.
Il modello più celebre in questo senso è il modello V-A (Valence-Arousal), nel quale le emozioni sono individuate su un piano bidimensionale che vede il parametro *arousal* (intensità) sull'asse verticale, e *valence* (polarità) su quello orizzontale. Alcuni modelli hanno proposto l'utilizzo di una terza dimensione [21], come ad esempio *tension, dominance* o *kinetics [22].*
L'individuazione delle emozioni in musica può in questo caso essere posto come un problema di regressione [23], affrontato ad esempio con tecniche quali la Support Vector Regression (SVR).

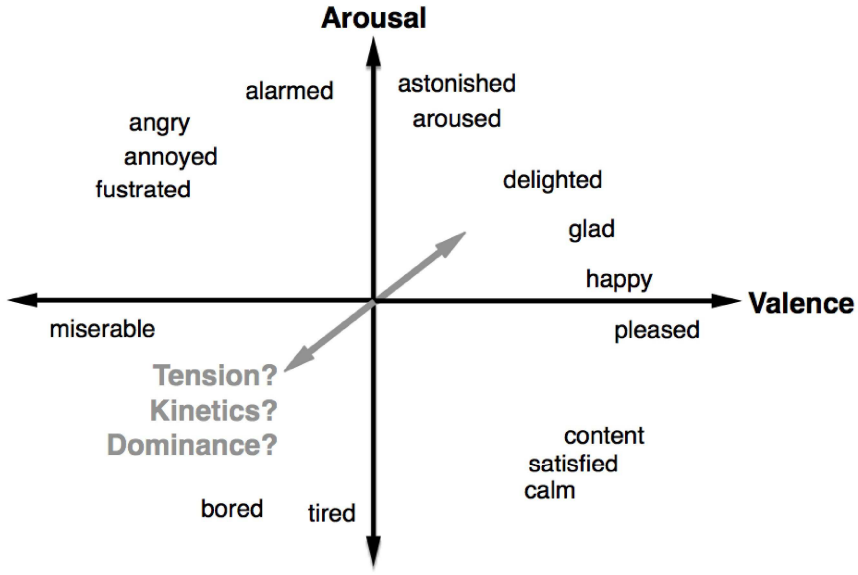


Fig 3.4: Piano Valence-Arousal nel modello delll'emozione di Thayer [17]

3.2 La visualizzazione del suono: esempi e progetti artistici

Oltre alle ricerche scientifiche, lo studio effettuato per questa tesi ha incluso anche aspetti artistici e grafici.
Un lavoro di fondamentale ispirazione è stato il libro Generative Design [24], che

descrive la creazione di grafiche interattive attraverso la programmazione in ambiente Processing e contiene un approfondito studio di design sull'utilizzo di forme e colori.

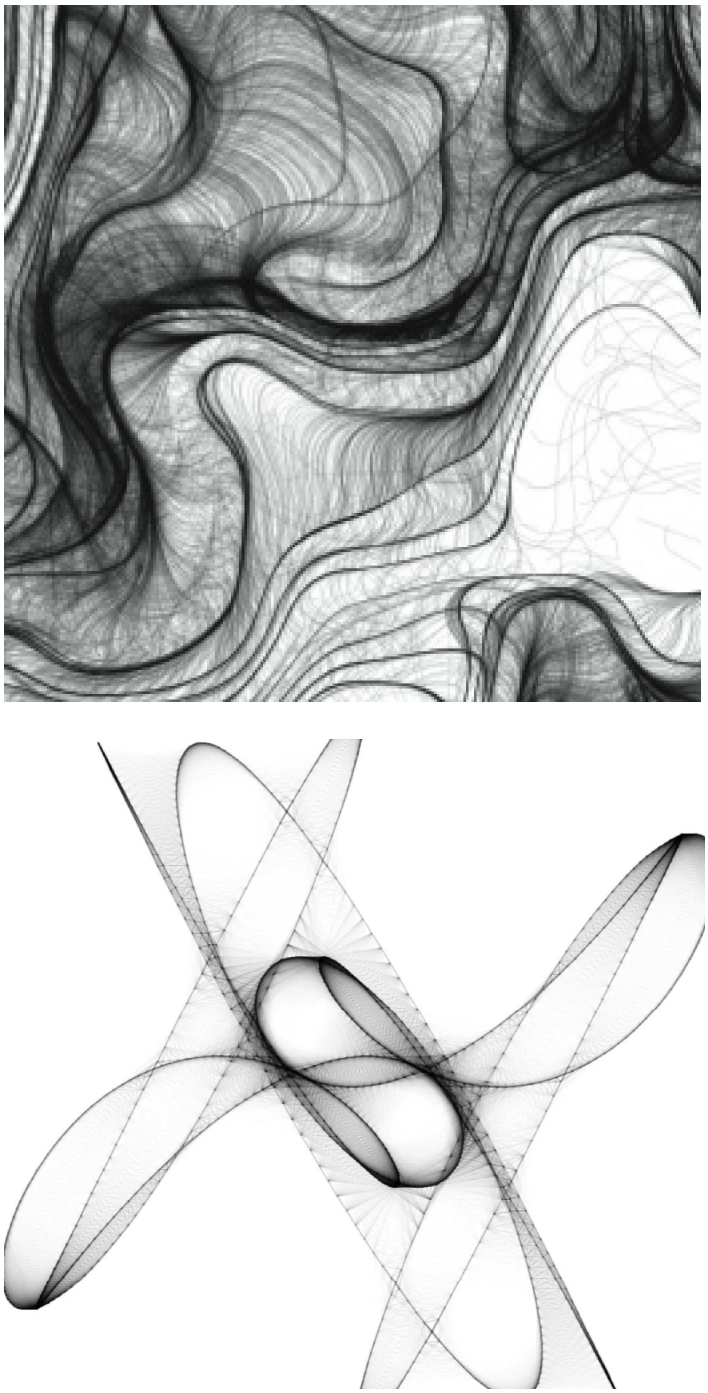


Fig.3.5: Fig. x: Immagini tratte dal libro Generative Design

Per quanto riguarda le grafiche reattive all'audio, come punto di partenza esistono numerosi software di riproduzione che forniscono visualizzatori musicali e generatori di effetti grafici che vengono solitamente sincronizzati con la musica, ma che non sono legati ad altri aspetti percettivi o emotivi.

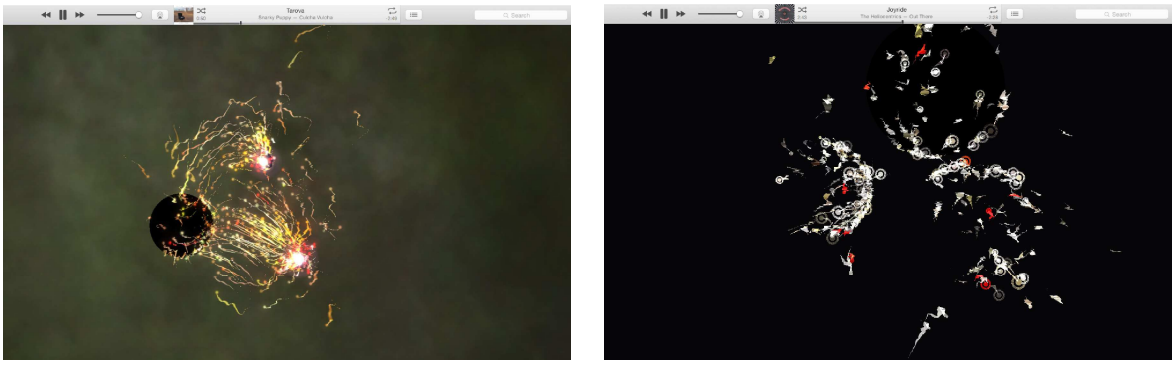


Fig. 3.6: Esempi di visualizzazioni integrate in iTunes

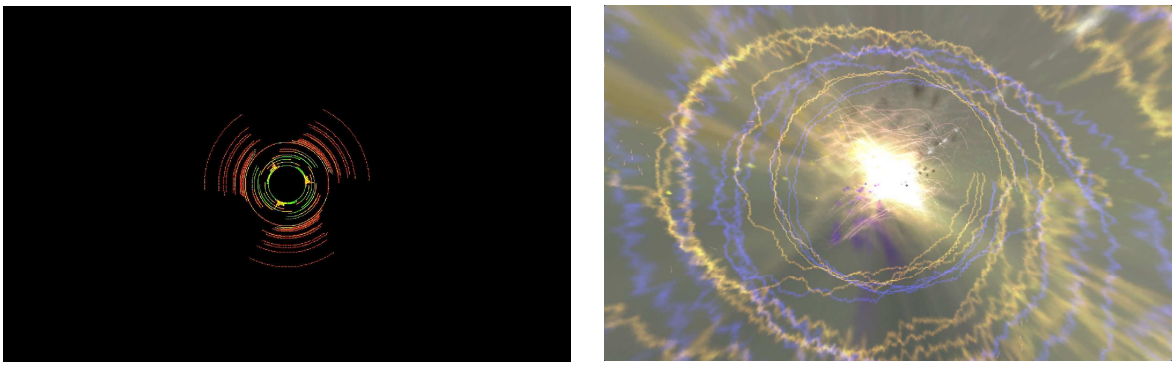


Fig. 3.7: Esempi di visualizzazioni integrate in VLC Media Player

Esistono studi che hanno tentato di generare una visualizzazione più rappresentativa della musica, o di alcuni aspetti di essa [25], e che si sono concentrati perlopiù su immagini fisse e non su immagini in movimento o animazioni [26,27].

Lavori più approfonditi in questo senso sono stati portati avanti con diversi approcci, sia per quanto riguarda il metodo di estrazione dei dati, la modellazione matematica dell'emozione, ma anche nelle scelte stilistiche e grafiche [28], e sono stati sperimentati sistemi per l'utilizzo durante performance live [29].

Facendo invece un passo indietro nel tempo non si può non citare Oskar Fischinger [30], regista tedesco considerato uno dei più importanti pionieri dell'animazione, nonché, nella prima metà degli anni '30, realizzatore di esperimenti di video astratti in cui metteva in relazione musica ed effetti grafici [31,32].

Egli, si ispirò a sua volta ad un altro importante regista tedesco, Walter Ruttman [33] che girò nel decennio precedente dei cortometraggi di animazione astratta [34,35], che furono un importante riferimento per Fischinger e per molti registi ed animatori a venire [36].

I lavori di questi due autori sono di straordinario spessore artistico e tecnologico,

soprattutto tenendo in considerazione l'epoca ed il contesto in cui sono stati sviluppati, e sono stati un'importante fonte di ispirazione e motivazione.

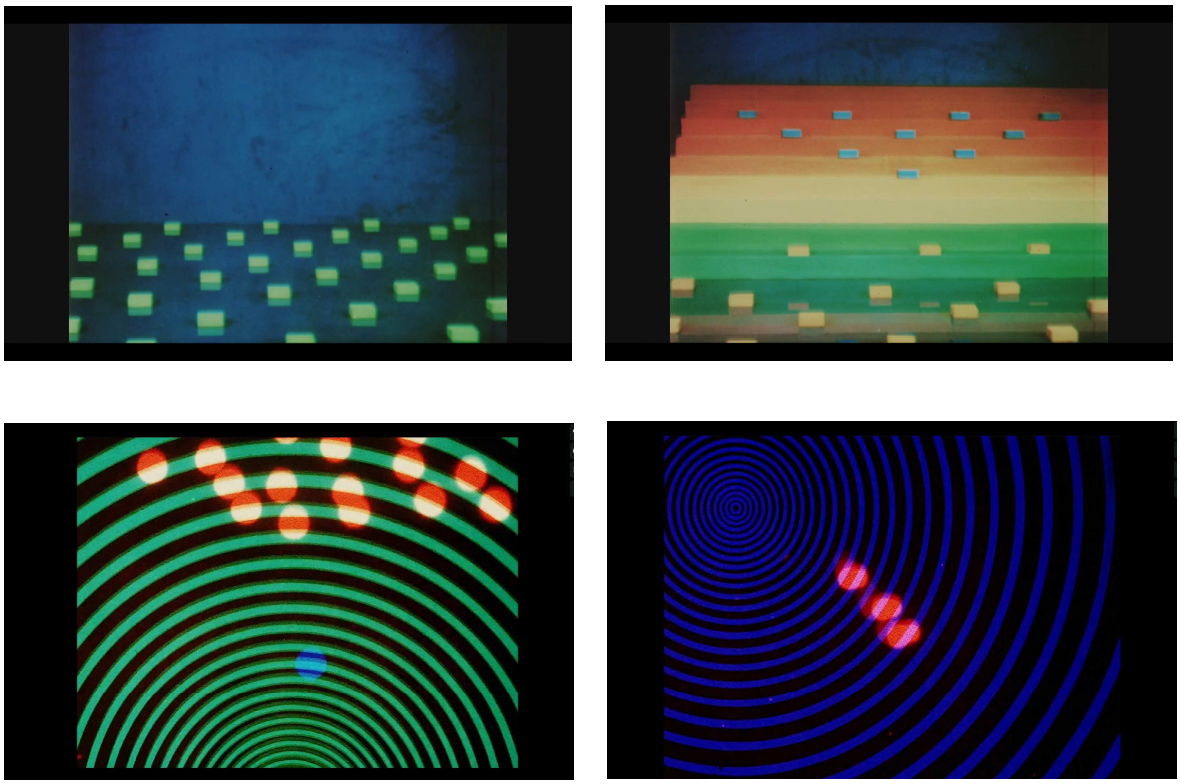


Fig. 3.8: Screenshots estratti da "Komposition in Blau" (sopra) e "Kreise" (sotto), entrambi di Fischinger

Tornando invece all'età contemporanea, di particolare influenza sono stati i lavori dell'artista Ryoichi Kurokawa, autore di numerose installazioni audio/visuali di grande raffinatezza ed impatto emotivo. [37]

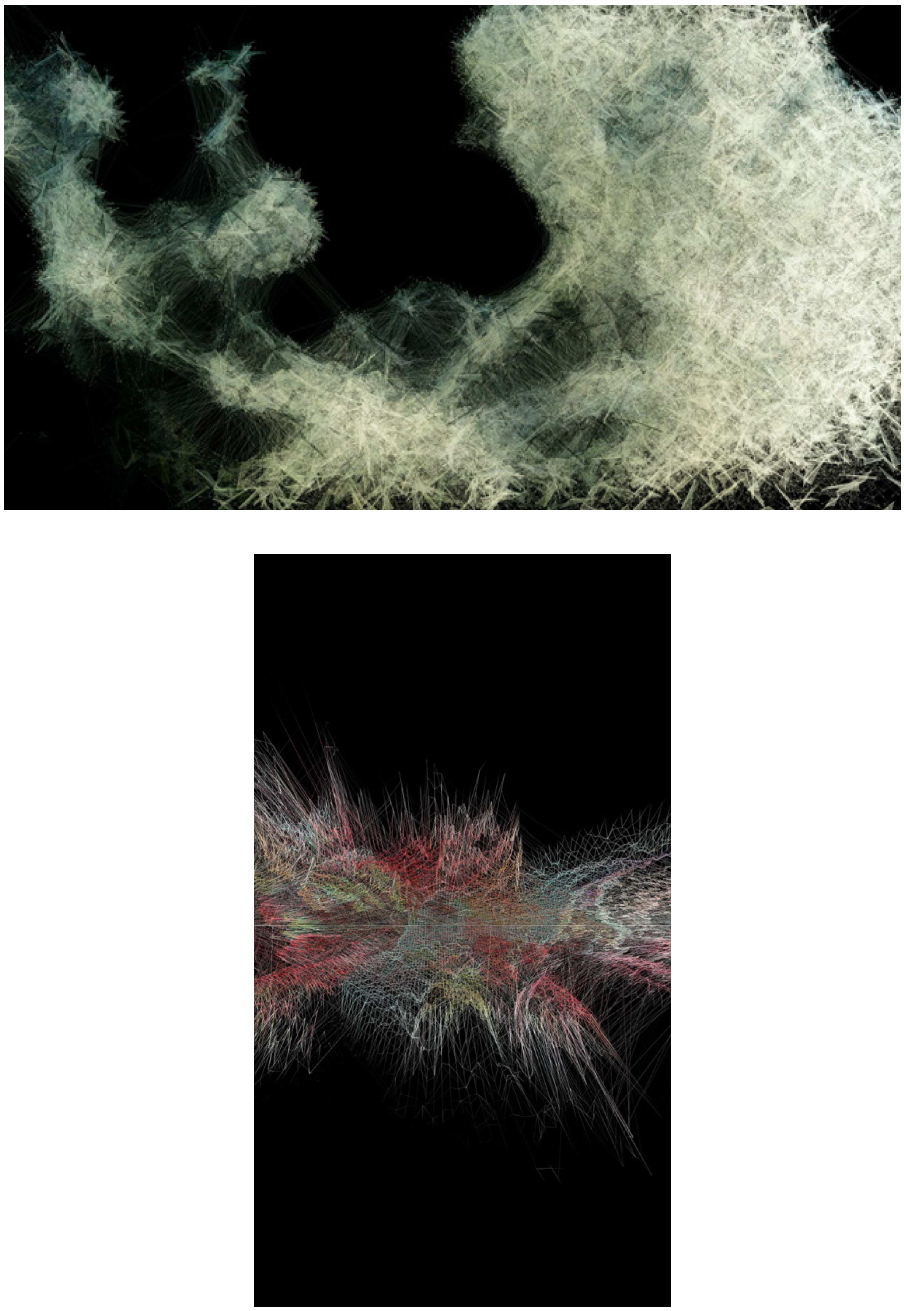


Fig. 3.9: Immagini tratte dal sito www.ryoichikurokawa.com

La visione di questo materiale ha fornito importanti spunti di riflessione in merito all'utilizzo di forme, colori e movimenti ed alle interazioni fra audio e video. Ricoprire in questo caso il ruolo di spettatori, ha permesso di toccare un punto di vista diverso, rivelatosi prezioso quando si è ricoperto, successivamente, il ruolo di "creatori". È in particolare emerso che la sincronizzazione degli effetti grafici con le transizioni video è di grande impatto sull'osservatore. Più agitata e dinamica è la musica, più mobili e reattive devono essere le transizioni.

3.3 Strumenti e tools per ambienti audio/visuali

Per comprendere al meglio il contesto in cui si colloca il dispositivo, sono state analizzate diverse proposte, sia hardware che software, già disponibili sul mercato con funzionalità analoghe a quelle di Dreamo.
Sono state analizzate in particolare due categorie di prodotti:

- Software dedicati allo sviluppo grafico e multimediale real-time
- Sintetizzatori video analogici

3.3.1 Software ed ambienti di sviluppo

Fanno parte della prima categoria gli ambienti di sviluppo orientati alla generazione e manipolazione real-time di contenuti multimediali.
In particolare il Max[5] è un software estremamente potente che, grazie alla sua struttura modulare, permette la programmazione di applicazioni musicali e multimediale di grande complessità. Max è un linguaggio di programmazione visuale e si basa su interfaccia a blocchi. Esistono diverse patch per questo ambiente di programmazione, fra cui MSP dedicata all'analisi e sintesi audio, e Jitter, dedicata alla gestione dei video. Queste patch forniscono numerose funzionalità disponibili sotto forma di "blocchi", i quali possono essere collegati e messi in relazione fra loro, creando così la struttura del programma.

5 Maggiori informazioni su https://cycling74.com/products/max/

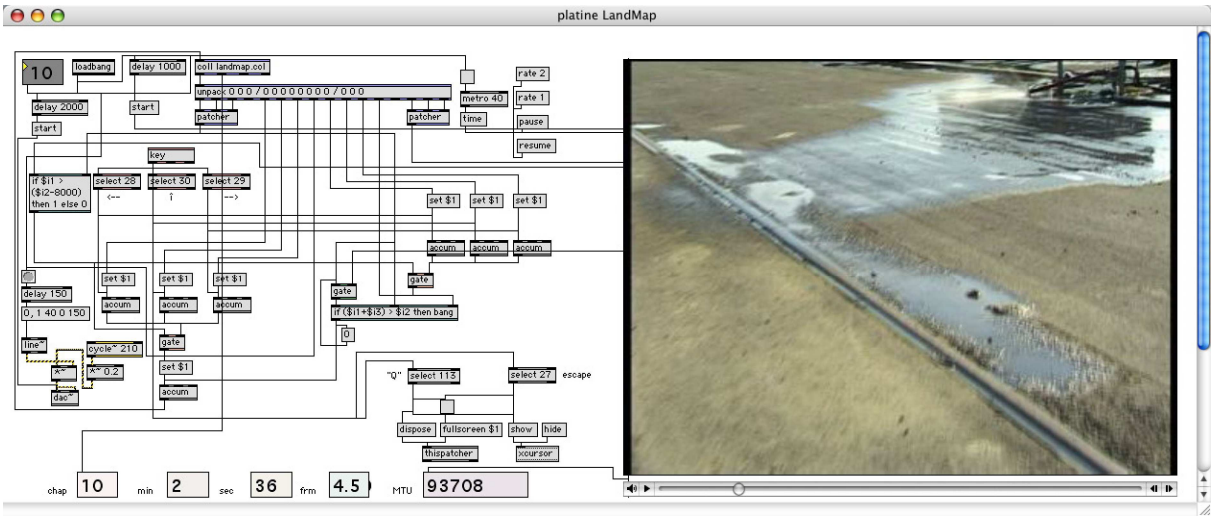


Fig. 3.10: Screenshot dell'interfaccia di una delle prime versioni di Max/MSP

Max è sicuramente uno dei software più completi sul mercato per questo tipo di utilizzo. È divenuto infatti negli ultimi anni lo standard di riferimento per tutti i performer e artisti di questo settore. Max è distribuito su licenza.

Un'alternativa free (almeno per scopi non commerciali) è VVVV[6], un ambiente sviluppo ibrido grafico/testuale molto versatile, particolarmente adatto alla generazione di video.

Questi ultimi due prodotti non presentano tuttavia nessuna funzionalità legata al riconoscimento delle emozioni, ed in generale la scrittura di applicazioni che comportano l'interazione fra audio e video necessitano spesso di generare diagrammi a blocchi molto complessi.

Un altro software famoso in questo ambito è Modul8[7], principalmente utilizzato da VJ, ed orientato al mix e compositing real time di video. Consente infatti di applicare effetti, generare transazioni ed di applicare numerose manipolazioni in tempo reale a video ed immagini.

3.3.2 Sintetizzatori video analogici

La seconda categoria di prodotti che abbiamo preso in analisi è quella dei sintetizzatori

6 Maggiori informazioni su https://vvvv.org
7 Maggiori informazioni su http://www.modul8.ch/

video analogici. Questi strumenti non hanno la potenza e la flessibilità degli ambienti di sviluppo software, ma grazie alla loro interfaccia fisica risultano molto adatti l'utilizzo durante performance o esibizioni dal vivo.

Esistono varie tipologie di sintetizzatori video analogici, con diverse funzionalità quali il compositing di video esterni, la generazione di segnali video attraverso oscillatori, il luma/chroma keying ecc...

In figura 3.11 e 3.12 sono presentati due esempi di quadd di questa categoria.

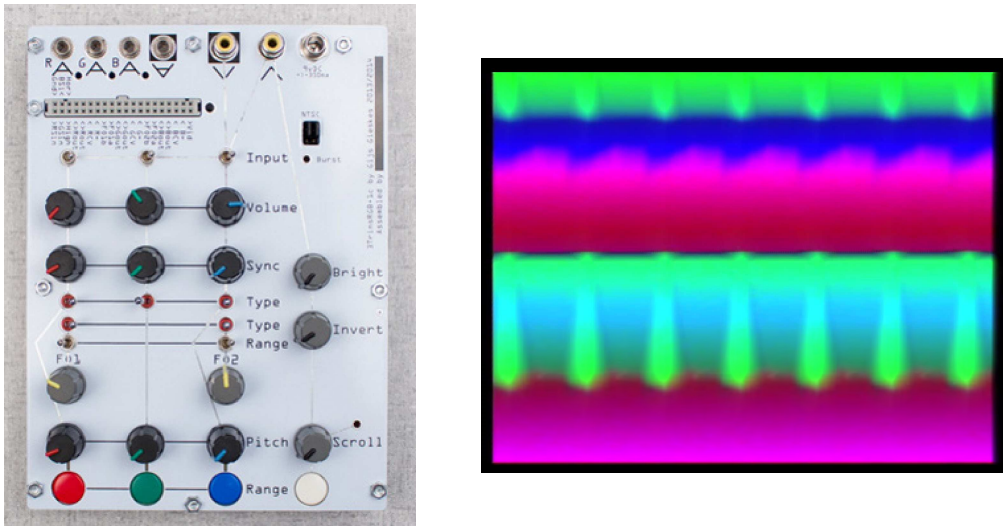


Fig. 3.11: Bleep Laps analog video synth

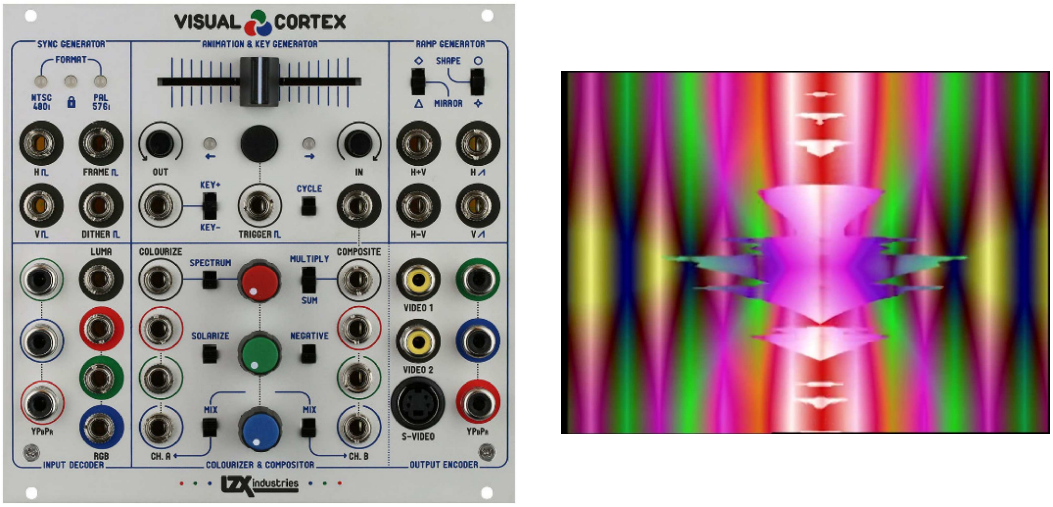


Fig. 3.12: LZX Visual Cortex

Questo tipo di strumenti, per quanto producano effetti video particolari ed interessanti, e per quanto siano più immediati e naturali nel loro utilizzo dal vivo, non sono paragonabili alle prestazioni e alla versatilità delle soluzioni digitali.

Da queste ricerche è emerso quindi che non esiste un vero e proprio strumento in grado analizzare l'audio dal punto di vista emotivo, e consentire un'interazione ad alto livello in tempo reale con un risultato grafico. Esistono alcune soluzioni che permettono di avvicinarsi a questo scopo ma che richiedono all'utente lo sviluppo di programmi non semplici da maneggiare e comprendere. Il progetto Dreamo intende quindi collocarsi in questo spazio, fornendo un nuovo approccio mood oriented all'interazione real-time di contenuti audio-visuali.

Capitolo 4

L’analisi dell’audio

In questo capitolo saranno esposte e giustificate le scelte tecniche prese durante lo sviluppo del software di analisi audio.
Sulla base dello studio della teoria, e dei lavori disponibili in letteratura, si è proceduto ad una scelta accurata delle features da estrarre e, non perdendo di vista le finalità del progetto, delle migliori tecniche con cui farlo. Nel primo paragrafo verrà esposto come sono state affrontate le problematiche legate alla natura *real-time* del progetto. Nel paragrafo 4.2 sarà invece descritta la struttura del software di estrazione e analisi dell’audio, mentre nel paragrafo successivo saranno descritte in dettaglio le scelte e le implementazioni delle varie tecniche di analisi. Il paragrafo 4.4 è dedicato all’esposizione di come vengono interpretati gli andamenti delle features estratti per la codifica di uno stato della traccia, mentre il sesto ed ultimo paragrafo illustra l’interfaccia utente implementata nel nostro sistema.

4.1 Un’applicazione in real time

Uno degli aspetti caratteristici di questo progetto è la sua natura *real-time*. Il motore grafico di Dreamo deve essere in grado di assumere comportamenti coerenti con l’andamento dell’audio, al fine di aumentare il coinvolgimento emotivo dello spettatore della performance. Questo ci ha fatto inevitabilmente propendere per un bassa complessità computazionale. La scelta è ricaduta quindi su features significative e semplici da calcolare.

Inoltre, l’utilizzo in tempo reale, rende molto difficile l’implementazione di tecniche di machine learning simili a quelle basate su modelli statistici affrontate nel paragrafo 3.1.3.
È stata quindi temporaneamente abbandonata l’idea di realizzare un vero e proprio *mood detector*, preferendo un sistema in grado di rilevare e monitorare l’andamento di features importanti e significative dal punto di vista percettivo e psicoacustico, utilizzabili poi da un sistema di rendering video, ottenendo così un video *responsive* all’audio.

Basandosi su principi psicologici ed estetici (come ad esempio quelli alla base dei lavori presentati nel paragrafo 3.2), e sul personale gusto artistico dei componenti del team, è stato creato un motore in grado di generare delle immagini in movimento coerenti all'andamento della musica, senza tuttavia che esso sia cosciente dell'esatto stato d'animo della traccia, calcolato secondo uno dei modelli matematici proposti in letteratura.

Resta comunque importante sottolineare che nel campo dell'arte ogni scelta è arbitraria e se fatta con coscienza non può considerarsi sbagliata e che questo sistema è uno strumento a disposizione di chiunque voglia cimentarsi nella sperimentazione di effetti ottenuti dall'interazione audio/video.

4.2 Il motore di analisi

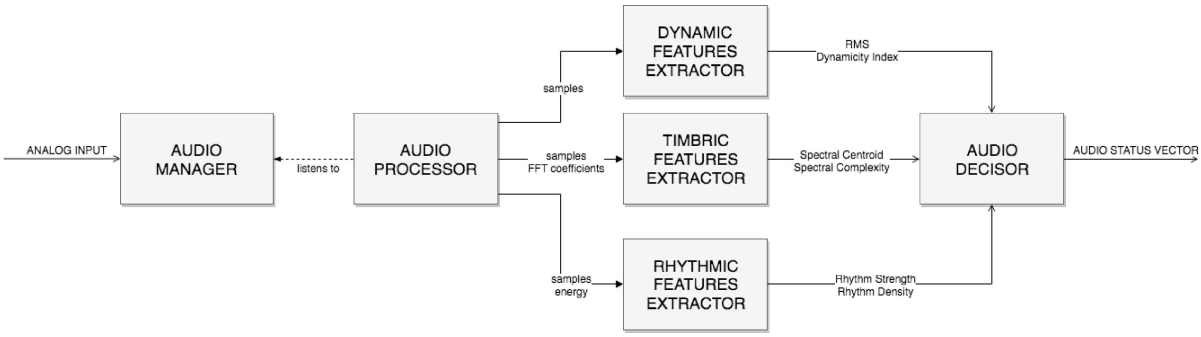


Fig. 4.1: Schema a blocchi del motore di analisi audio

Il motore di analisi audio si divide in 4 livelli, individuabili da altrettante classi java:

- **Audio Manager**: sfrutta i metodi della libreria Minim[8] (come illustrato precedentemente) e si occupa di determinare la frequenza di campionamento, bufferizzare i campioni in ingresso e determinare il tipo di ingresso (stream/traccia offline, mono/stereo). Attualmente l'audio viene acquisito con una frequenza di campionamento di 44100 Hz e un buffer di 1024 campioni.

- **Audio Processor:** implementa l'interfaccia AudioListener presente nella libreria Minim, e possiede due metodi che vengono richiamati non appena l'istanza

8 codice disponibile su https://github.com/ddf/Minim

di AudioManager ha a disposizione un buffer di nuovi campioni. Si occupa di effettuare le operazioni comuni come il calcolo dell'FFT, il calcolo dell'energia media di ogni frame, ecc…

- **Features Extractors**: si occupano del calcolo specifico delle features acustiche. Sono divisi per categoria e saranno esposti in dettaglio nel paragrafo successivo.

- **Audio Decisor:** sulla base delle features estratte calcola un vettore di "stato" rappresentativo dell'audio e fornisce dei parametri che aggregano lo stato di due o più features acustiche. Rileva inoltre se ci sono cambi o transizioni nello stato della traccia. È descritto in dettaglio nel paragrafo 4.5.

4.3 La scelta delle features

Una parte importante di lavoro è stata dedicata alla scelta dei parametri audio da estrarre. Le emozioni dell'ascoltatore sono direttamente influenzate da attributi acustici quali timbro, ritmo, intensità e tonalità (più specificatamente modalità). È condiviso che non esista una caratteristica acustica "dominante" per quanto riguarda la determinazione del mood, e per questo bisogna orientarsi su un set di features adeguate al task e alla potenza computazionale disponibile [17].
Gli esempi in letteratura sono molti e comprendono caratteristiche dinamiche (energia, …), timbriche (MFCC, features spettrali, …) ritmiche (bpm, regolarità…), armoniche (tonalità, cromagrammi, ...) o di altra natura [17] [19] [18] [38] [4] [23].
La nostra scelta è ricaduta infine su 6 features, separabili in 3 categorie, in base al dominio di estrazione.

4.3.1 Features dinamiche

Uno degli aspetti più immediati dal punto di vista percettivo, è la dinamica del suono. Suono forti o suoni deboli influiscono notevolmente sull'ascoltatore, e determinano in maniera fondamentale come viene percepito un brano musicale [19].

RMS

L'RMS è una buona misura della potenza del segnale, ed è calcolato sommando i quadrati di ogni campione, dividendo tale somma per il numero di campioni della la finestra di osservazione (la lunghezza del buffer) ed estraendo la radice quadrata del risultato [39].

$$RMS = \sqrt{\frac{\sum_{i=0}^{N-1} s[i]^2}{N}}$$

Dove N è il numero di campioni (quindi la lunghezza del buffer, 1024 nel nostro caso), e s[i] rappresenta l'i-esimo campione.

DYNAMICITY INDEX

Un altro aspetto importante è la variazione di volume, che viene preso in considerazione tenendo traccia della deviazione standard dell'RMS.

$$Dynamicity\ Index = \sqrt{\frac{\sum_{i=0}^{N-1}(RMS_i - \overline{RMS})^2}{N-1}}$$

Dove N rappresenta il numero unità statistiche prese in analisi (nel nostro caso 129 valori di RMS, quindi circa 3 secondi di audio) e $\overline{RMS}$ rappresenta il valore medio calcolato sullo stesso intervallo di tempo.

Tale misura, chiamata Dynamicity Index, rappresenta quindi la variabilità del segnale in una finestra di 3 secondi, e corrispode alla deviazione stanrard dell'RMS calocata in tale intervallo.

L'estrazione delle features dinamiche è affidata alla classe *dynamic* che estende la classe feature extractor.

4.3.2 Features timbriche

Le features timbriche sono numerose e hanno un ruolo molto importante nella

modellazione. In linea con la natura di questo lavoro sono state scelte features con un basso costo computazionale e con una grande rilevanza dal punto di vista percettivo.

CENTROIDE SPETTRALE

Rappresenta il baricentro dell spettro. È percettivamente legato alla "brillantezza" del suono ed è una caratteristica importante, rappresentativa del timbro di un brano musicale. [40] [18] [15]

$$Centroid = \frac{\sum_{n=0}^{N-1} f(n)x(n)}{\sum_{n=0}^{N-1} x(n)}$$

Dove N è il numero di bin dell'FFT, x(n) rappresenta l'ampiezza dell'n-esimo bin dell'FFT e f(n) rappresenta la frequenza centrale dell'n-esimo bin.

COMPLESSITÀ SPETTRALE

La complessità spettrale è un descrittore basato sul numero di picchi nello spettro. Viene quindi applicato un algoritmo di *peak detection* alla allo spettro in uscita dal'FFT, e si conta il numero di picchi [18], che superano una soglia di rumore.

ZERO CROSSING RATE

Rappresenta il numero di passaggi dallo zero del segnale audio. È un dato importante ed è alla base di diverse tecniche di analisi dell'audio [41]. Per i segnali monotonali è la base di tecniche di *pitch tracking [8]* ed è in grado di indicare il livello di "rumorosità" di un suono [42].

$$ZCR = \frac{1}{T-1}\sum_{t=1}^{T} \mathbb{I}\{s_t s_{t-1} < 0\}$$

Dove T rappresenta la lunghezza del segnale (in questo caso la dimensione del buffer) e I{A una mappa che vale 1 se A una proposizione vera, e 0 altrimenti.

Dopo aver implementato tali algoritmi inclusi nella classe *timbre*, sono stati effettuati diversi test su alcune tracce musicali. E' di seguito riportato il grafico dell'andamento delle features timbriche estratte dalla canzone "Angel", pubblicato nel 1998 dalla band inglese Massive Attack, con la propria forma d'onda e spettrogramma. L'andamento di tutte le features di questo brano è consultabile alla sezione A.6 dell'Appendice A.

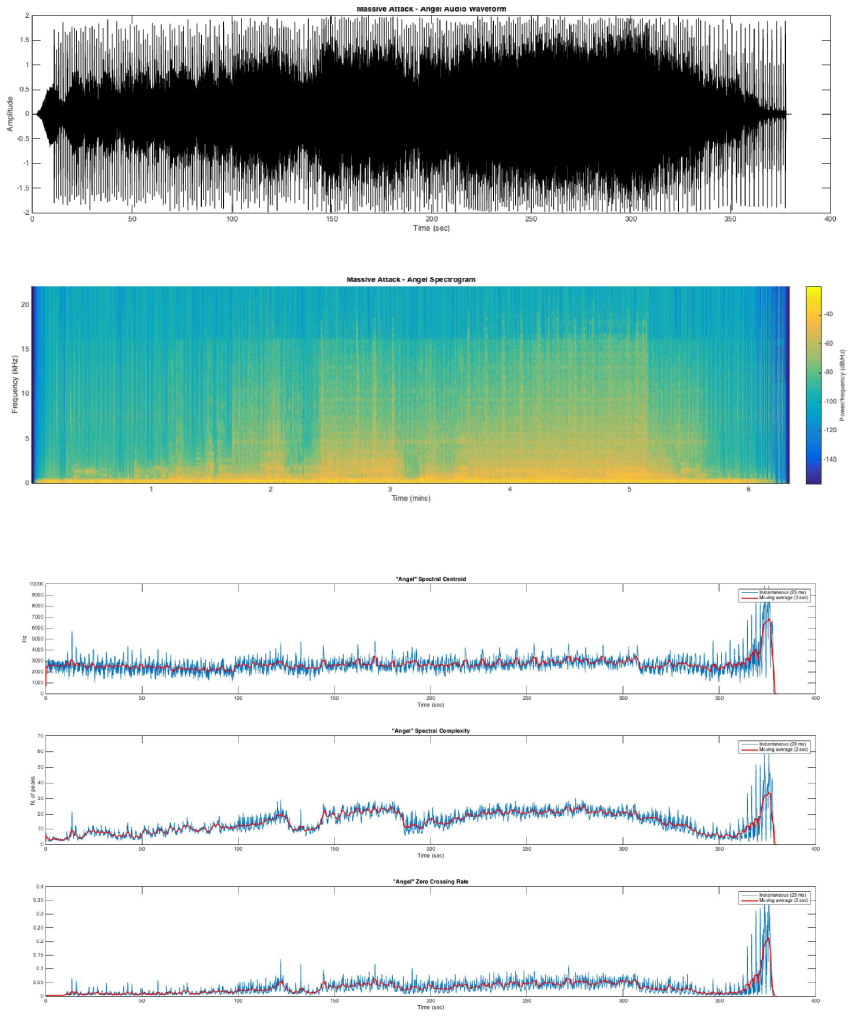


Fig. 4.2: Forma d'onda e spettrogramma (sopra) e andamento delle features timbriche (sotto) del brano "Angel" dei Massive Attack

Lo ZCR sembra seguire l'andamento della complessità spettrale. Questo parametro è stato quindi escluso dal set di features considerate dal sistema di decisioni presentato in questa tesi (ed esposto nel capitolo 5) perché sembra essere ridondante. Ciò non esclude il suo utilizzo futuro, soprattutto in previsione dell'implementazione di algoritmi di *key* e *chord detection*.

4.3.3 Features ritmiche

Il ritmo è uno degli aspetti fondamentali della musica e numerosi algoritmi di *beat-detection* sono stati proposti in letteratura. [19] Diversi sono stati gli approcci proposti: basati sulla DWT [43] sull'analisi di suoni percussivi [44].

Il calcolo esatto dei bpm[9] resta comunque un taks difficile da realizzare con precisione, soprattutto se eseguito in real-time, contesto nel quale molti degli algoritmi proposti non sono utilizzabili.

Era comunque necessario avere un'indicazione del contenuto ritmico di un brano musicale: il lavoro si è orientato quindi sul rilevamento degli onset, ovvero dei picchi di energia con un rapido attacco, che sono nella maggior parte dei casi dovuti a strumenti come (batterie o percussioni) e chiaramente legati alla percezione del ritmo.

È stato quindi implementato un onset detector basato sull'energia dei picchi nel dominio temporale: viene rilevato un onset ogni qualvolta il segnale presenta un picco di energiache supera una soglia definita come:

$$threshold = C * avg(E)$$

Dove

$$avg(E) = \frac{1}{43}\sum_{j=0}^{42} E_j$$

Con

$$E_j = \sum_{i=0}^{N-1} left[i]^2 + right[i]^2$$

dove N è pari al numero di campioni in un buffer. Inoltre

9 Beats per minute

$$C = -0.0000015 * var(E) + 1.5142857$$

dove

$$var(E) = \frac{1}{43}\sum_{j=0}^{42}(avg(E) - E_j)^2$$

Questo semplice approccio consente di avere un algoritmo leggero che, seppur non essendo in grado di fornire un valore esatto dei bpm o di riconoscere pattern regolari, fornisce un'indicazione sul contenuto ritmico. [45]
Vengono successivamente estratte due features ritmiche:

- rhythm strength;
- rhythm density.

RHYTHM STRENGTH

Rappresenta l'energia media degli onsets rilevati su un intervallo temporale si 2 secondi: durante questa finestra un contatore tiene traccia del numero di onset rilevati, mentre un sommatore ne accumula l'energia.
Trascorsi i due secondi il parametro *rhythm strength* viene calcolato come:

$$rhythm\ strength = \frac{tot\ onsets\ energy_{2\ sec}}{onsets\ number_{2\ sec}}$$

RHYTHM DENSITY

Rappresenta la densità di onsets in un secondo. A partire dagli onsets rilevati nella finiestra di 2 secondi, il descrittore *rhythm density* viene calcolato come segue:

$$rhythm\ density = \frac{onsets\ number_{2\ sec}}{2}$$

Allo scopo di verificare il funzionamento degli algoritmi di estrazione delle features ritmiche, implementati come metodi della classe *rhythm* è stata generata una traccia audio di test[10] che si sviluppa in tre fasi. Di seguito sono riportati i grafici relativi all'analisi del brano.

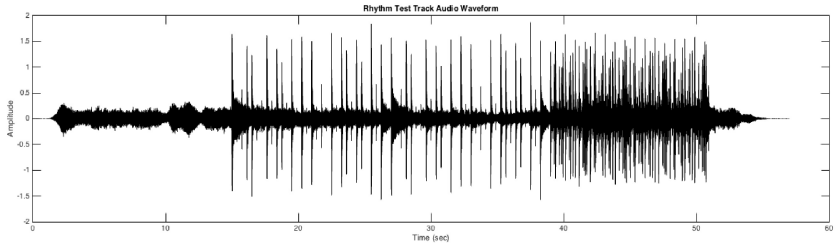


Fig. 4.3: Forma d'onda della traccia di test

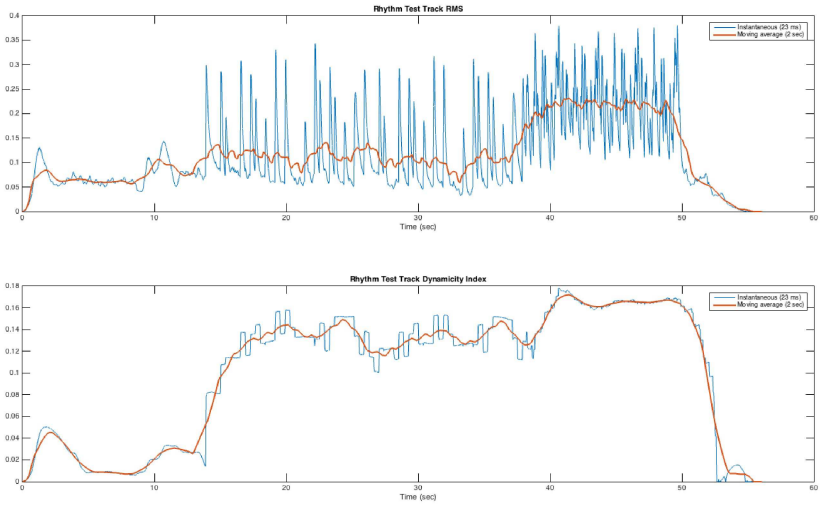


Fig. 4.4: Grafico dell'andamento delle features dinamiche della traccia di test

10 La traccia è ascoltabile al seguente link: https://soundcloud.com/user-13688703/dreamo-rhythmic-test-track/s-XxNVY

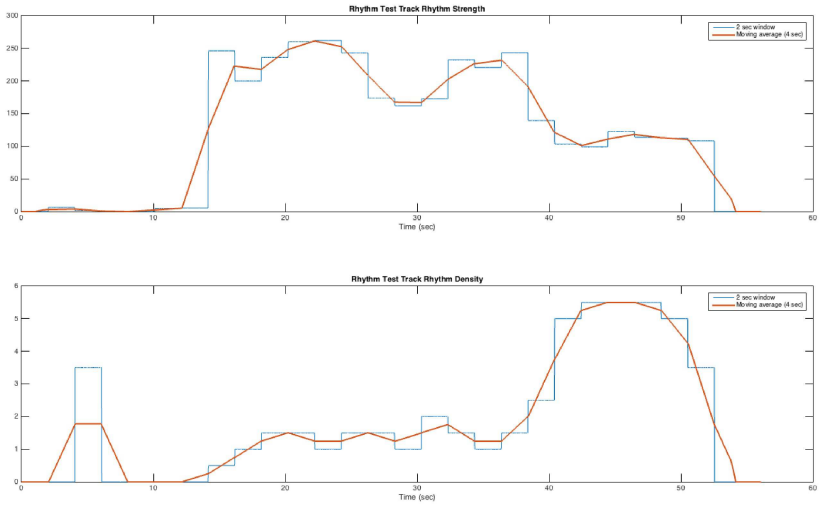


Fig. 4.5: Grafico dell'andamento delle features ritmiche della traccia di test

Come si può notare dalla forma d'onda (Fig. 4.3) la traccia presenta un'introduzione priva di contenuto ritmico e con un volume costante. La seconda fase (a partire da 15 secondi circa) presenta un ritmo lento e molto marcato. La terza ed ultima fase (a partire da 39 secondi circa) presenta invece un ritmo più veloce ma con picchi meno prominenti rispetto al volume medio della traccia in quella porzione di brano.

Si può quindi osservare dal grafico in Figura 4.5 che il valore del parametro *rhythm* strength cresce notevolmente nel momento in cui la prima percussione fa il suo ingresso, per poi calare leggermente dal secondo 39 in poi, quando subentra il ritmo più veloce ma più leggero. Coerentemente, il valore del parametro rhythm density cresce all'aumentare della velocità del ritmo.

Il picco visibile del grafico della rhythm density intorno al secondo 5 è dovuto al fatto che l'algoritmo si deve ancora stabilizzare calcolando la soglia per il rilevamento degli onsets. Nei primi secondi del brano tale soglia è ancora troppo bassa ed anche leggere oscillazioni nel volume possono causare un rilevamento di falsi positivi.

Nel grafico in Figura 4.4 si nota anche che all'aumentare nelle variazioni nel volume RMS aumenta il valore del parametro Dynamicity Index.

Nella tabella seguente sono riassunte le features estratte dal motore di analisi di Dreamo, con la loro tipologia e definizione: Nell'Appendice A sono raccolti i dati ricavati dall'analisi di alcuni brani utilizzati come test set.

FEATURE	TIPOLOGIA	DEFINIZIONE
RMS	dinamica	Valore efficace ***(root mean square)*** dei campioni del segnale calcolato su frames di 23 ms.
Dynamicity Index	dinamica	Deviazione standard dell'RMS calcolata ogni 23 ms su un intervallo di 3 secondi precedenti
Spectral Centroid	timbrica	Centro di massa dello spettro calcolata su frames di 23 ms.
Spectral Complexity	timbrica	Numero di picchi nello spettro calcolati su frames di 23 ms.
Rhythm Strength	ritmica	Energia media degli onsets rilevati in una finestra di 2 secondi
Rhythm Density	ritmica	Numero di onsets per secondo. Calcolato ogni 2 secondi.

Tabella 4.1: Set di features estratte dal motore di analisi

4.4 Lo "stato" della traccia

Dopo aver definito ed implementato gli algoritmi di estrazione delle features audio, è stato deciso di sfruttare tali dati per definire uno "stato" della traccia: per ognuno dei 6 descrittori disponibili sono stati fatti dei test con brani di diverso genere, stile e *mood*, e sono stati individuati degli intervalli di valori entro i quali lo stato viene codificato con tre possibili condizioni: basso, medio o alto.

Per definire meglio questi intervalli sono state inoltre calcolate le funzioni di densità di probabilità di ogni feature, utilizzando inoltre una seconda traccia di test originale, creata per questo scopo: questa traccia, chiamata "Dreamo Track", composta e prodotta

appositamente, contiene diverse fasi e atmosfere (assenza o presenza di ritmo, utilizzo di diversi strumenti, cambi di dinamica…), ed è stata di fondamentale importanza per definire e settare i parametri dell'algoritmo[11]. In Figura 4.6 sono presentati i grafici delle PDF di ogni feature della traccia "Dreamo Track", mentre i grafici dell'andamento temporale delle sue features sono disponibili alla sezione A.11 della'Appendice A. Inoltre i grafici delle PDF di tutte le tracce del test set sono disponibili nell'Appendice B.

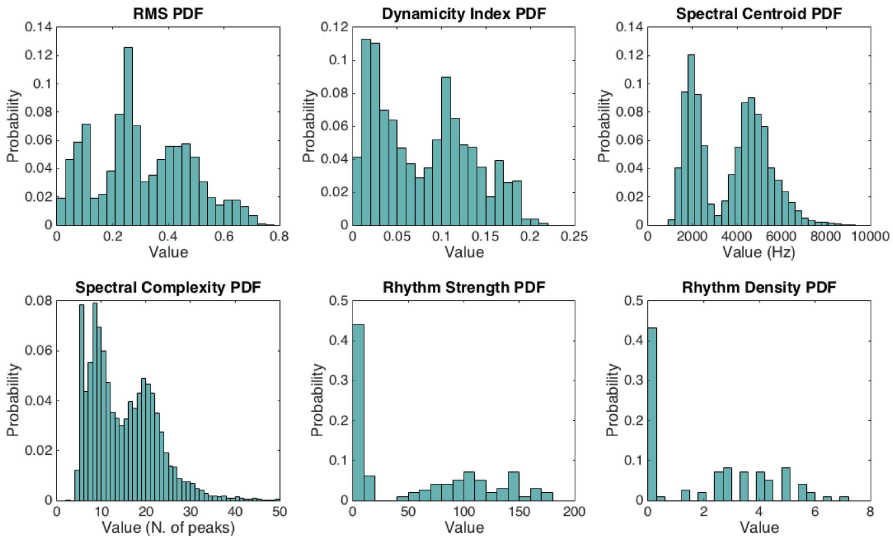


Fig. 4.6: Grafici delle PDF delle features estratte dalla traccia Dreamo Track

I valori di soglia sono riportati nella tabella 4.2, insieme all'intervallo in cui la feature viene osservata per la raccolta delle statistiche:
È infatti il valore medio, calcolato su un intervallo di tempo dell'ordine di qualche secondo (come da tabella), a determinare se lo stato corrente è alto, medio o basso. È stato inoltre aggiunto un controllo di isteresi per evitare oscillazioni continue nel valore dello stato.

11 La traccia è ascoltabile al seguente link: https://soundcloud.com/user-13688703/dreamo-track/s-smSeb:

FEATURE	FINESTRA OSSERVAZIONE	INTERVALLI STATI
RMS	2 secondi	Low < 0.3 Medium 0.3 - 0 .56 High > 0.56
Dynamicity index	2 secondi	Low < 0.1 Medium 0.1 - 0.19 High > 0.19
Spectral centroid [Hz]	2 secondi	Low < 2050 Medium 2050 - 3000 High > 3000
Spectral complexity [n. of peaks]	2 secondi	Low < 11 Medium 11- 18 High > 18
Rhythm strength	4 secondi	Low < 27.5 Medium 27.5 - 125 High > 125
Rhythm density [onsets/second]	4 secondi	Low < 1.5 Medium 1 - 4 High > 4

Tabella 4.2: Soglie per il rilevamento dello stato

Lo stato viene codificato numericamente in questo modo:

- BASSO=0;
- MEDIO=1;
- ALTO=3;

Questa codifica asimmetrica è stata scelta per pura comodità nel rilevamento dei cambi di stato: in questo modo con la sola differenza fra stato corrente e stato precedente è possibile capire la direzione del cambio ed il nuovo stato.

4.5 L'interfaccia utente

È stata implementata un'interfaccia utente che permette di controllare i valori delle soglie in tempo reale. In questo modo, se desiderato, l'utente può modificare la mappatura degli stati delle features, facendo variare radicalmente il comportamento del video, a causa dei meccanismi che saranno illustrati nel capitolo successivo.

Questa interfaccia è in linea con la filosofia "live" della performance nella quale il

progetto Dreamo vede la sua principale collocazione.

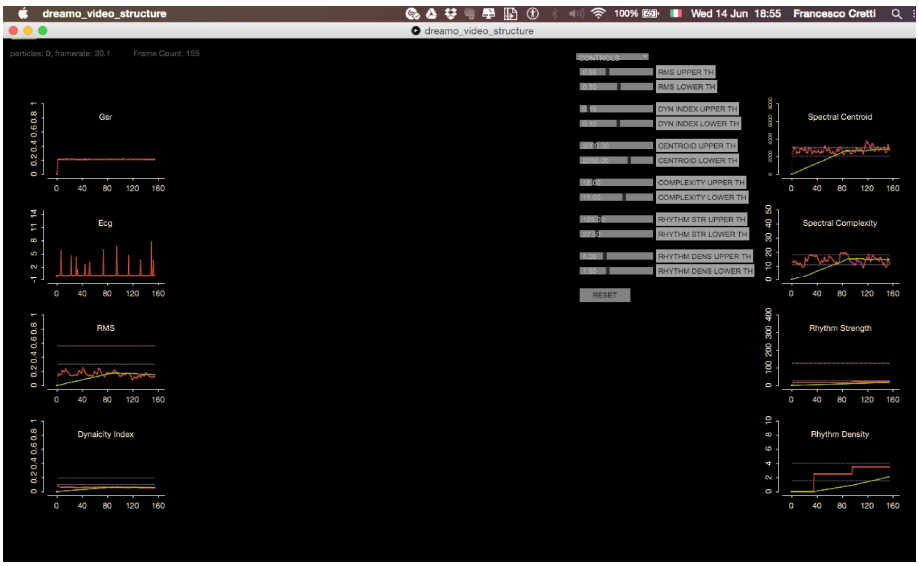


Fig. 4.7: L'interfaccia utente consente di modificare le soglie in tempo reale

Capitolo 5

Interazioni audio/video

Il seguente capitolo è dedicato alla presentazione dei meccanismi d'interazione tra audio e video e delle funzionalità che sono state implementate nel nostro sistema, dando un'idea del loro possibile utilizzo.
Dopo aver osservato diverse realizzazioni di opere ed installazioni legate alle interazioni fra audio e video, si è giunti alla conclusione che alcuni aspetti sono di fondamentale importanza per il collegamento dell'esperienza uditiva a quella visiva da parte dell'ascoltatore. Inoltre, sono stati tenuti in forte considerazione gli aspetti legati al messaggio emotivo espresso dalla musica, in linea con lo scopo originale del progetto. Sulla base di questo, una realizzazione di un effetto grafico reattivo all'audio è esposta nel paragrafo 5.2.
Dreamo vuole essere tuttavia uno strumento versatile, qualsiasi sia l'idea di interazione audio/video dell'utilizzatore. A supporto di questo, una breve guida su come creare un nuovo effetto grafico è presente nella sezione 5.3.

5.1 Interazioni *mood oriented*

Era a questo punto importante capire l'andamento dei parametri acustici condizionare la percezione delle emozioni nell'ascoltatore. Che caratteristiche ha una canzone felice? Come si evolvono le features di un brano triste? Che differenza c'è nel profilo acustico fra musica rilassata e tesa? Solo dopo aver ragionato su questi aspetti e compreso come il nostro sistema poteva sfruttarli, è stata realizzata una prima grafica *mood oriented*, sensibile ai cambiamenti dell'audio.

Come descritto precedentemente, un aspetto preponderante nella percezione della musica è quello dinamico: il volume di una traccia è il primo ed immediato riferimento uditivo che giunge all'ascoltatore. È quindi molto importante che i movimenti delle immagini siano sincronizzati con le variazioni di volume del brano musicale. È possibile infatti affermare che l'intensità (modellata con l'RMS) è direttamente correlata con la

percezione dell'energia [19].

Un secondo aspetto è il timbro: timbri aperti e chiari trasmettono sensazioni molto diverse da timbri scuri o *smooth*. Come citato in precedenza è possibile assumere che timbri con frequenza centroide e complessità più alte siano associati a sensazioni di energia, o stress, in base alla combinazione delle altre *features* acustiche [19]. Suoni con un profilo armonico più semplice hanno timbri percettivamente più "scuri" e tendono a "placare" le emozioni umane [42], non a caso brani etichettati come "rilassati" tendono ad avere complessità spettrali in media minori di quelli "non-rilassati". D'altro canto un brano "felice" tende ad avere una complessità spettrale maggiore di uno "non-felice". [18].

Il ritmo ha un forte rilevanza nella determinazione del mood [15,19]. Tempo (velocità del ritmo), energia, regolarità, metrica sono elementi rappresentativi del ritmo, distintivi di generi musicali provenienti da culture e contesti sociali diversi [46]. Combinazioni diverse di questi parametri hanno un effetto totalmente diverso sull'ascoltatore. Il ritmo è rappresentato nel nostro sistema dai parametri *rhythm strength* e *rhythm density*, che rappresentano l'energia e la densità degli onset. Possiamo quindi in generale assumere che più sono alti questi parametri più è alto il grado di eccitazione del brano in questione [18].

Il discorso sul pitch è più complesso: quando si analizza una traccia composta dall'unione di molti strumenti, è difficile stabilire da un punto di vista percettivo, quale sia il pitch preponderante per l'ascoltatore. L'unione delle note suonate dai diversi strumenti, contribuisce infatti alla creazione dell'armonia[12] che risulta essere una caratteristica fondamentale della canzone. Sarebbe quindi più sensato puntare al rilevamento della modalità musicale (intesa come armonia maggiore o minore), sebbene sia un *task* di difficile da raggiungere. Esistono studi al riguardo [47] e proposte di algoritmi utili per questo scopo [48] [8], e questo lavoro è attualmente in fase di sviluppo.

La Figura 5.1 presenta una tabella tratta da [42] che riassume, secondo gli studi condotti dagli autori, il legame qualitativo fra features acustiche ed emozioni.

12 Intesa del punto di vista musicale, come un insieme di note che suonano contemporaneamente

Mood	Intensity	Timbre	Pitch	Rhythm
Happy	Medium	Medium	Very High	Very High
Exuberant	High	Medium	High	High
Energetic	Very High	Medium	Medium	High
Frantic	High	Very High	Low	Very High
Anxious/Sad	Medium	Very Low	Very Low	Low
Depression	Low	Low	Low	Low
Calm	Very Low	Very Low	Medium	Very Low
Contentment	Low	Low	High	Low

Fig. 5.1: Correlazione tra caratteristiche acustiche ed emozione espressa dalla musica tratto da [42]

A livello percettivo, sono inoltre fondamentali i cambi all'interno della traccia: quando uno strumento comincia o termina la propria parte l'ascoltatore percepisce questo cambiamento, e conferisce maggiore attenzione a ciò che sta succedendo all'interno della traccia.

Il nostro sistema sfrutta quest'ultimo aspetto per creare grafiche coerenti: nel momento in cui viene rilevato un cambiamento all'interno della traccia, inteso come cambiamento di stato di una o più feature, il sistema deve reagire dando così un feedback immediato allo spettatore.

5.2 Realizzazione di un video *responsive*

Analizzando gli aspetti illustrati nel paragrafo precedente, si è proceduto impostando due livelli di interazione:

- **assegnazioni dirette:** un parametro grafico viene controllato direttamente da un parametro acustico (ad esempio l'RMS controlla direttamente la posizione, la centroide spettrale controlla la saturazione di un colore, ecc…). Consentono di avere un feedback immediato rispetto all'andamento della traccia. Per alcuni di questi collegamenti può essere necessario rimappare od adattare la scala dei valori.

- **decisioni in base allo stato:** il rilevamento dei cambiamenti dello stato consente di far reagire il sistema coerentemente con transizioni importanti nella traccia. Calcolando infatti quali e quante features variano, e come variano, è possibile scatenare delle reazioni, prendendo così decisioni di "alto livello" che cambiano il modo di interazione fra audio e video.

5.2.1 Assegnazioni dirette

Primi test sono stati svolti con lo scopo di testare quali parametri audio fossero più adatti per il controllo il video, e quali collegamenti impostare fossero i più efficaci.

- **Esempio 1: effetto del volume sulla velocità** Come primo esperimento è stata creata una scena composta da una griglia di particelle circolari ed è stato collegato il volume RMS alla posizione delle particelle stesse. La direzione del movimento è invece scelta randomicamente. Di seguito qualche screenshot dei risultati ottenuti:

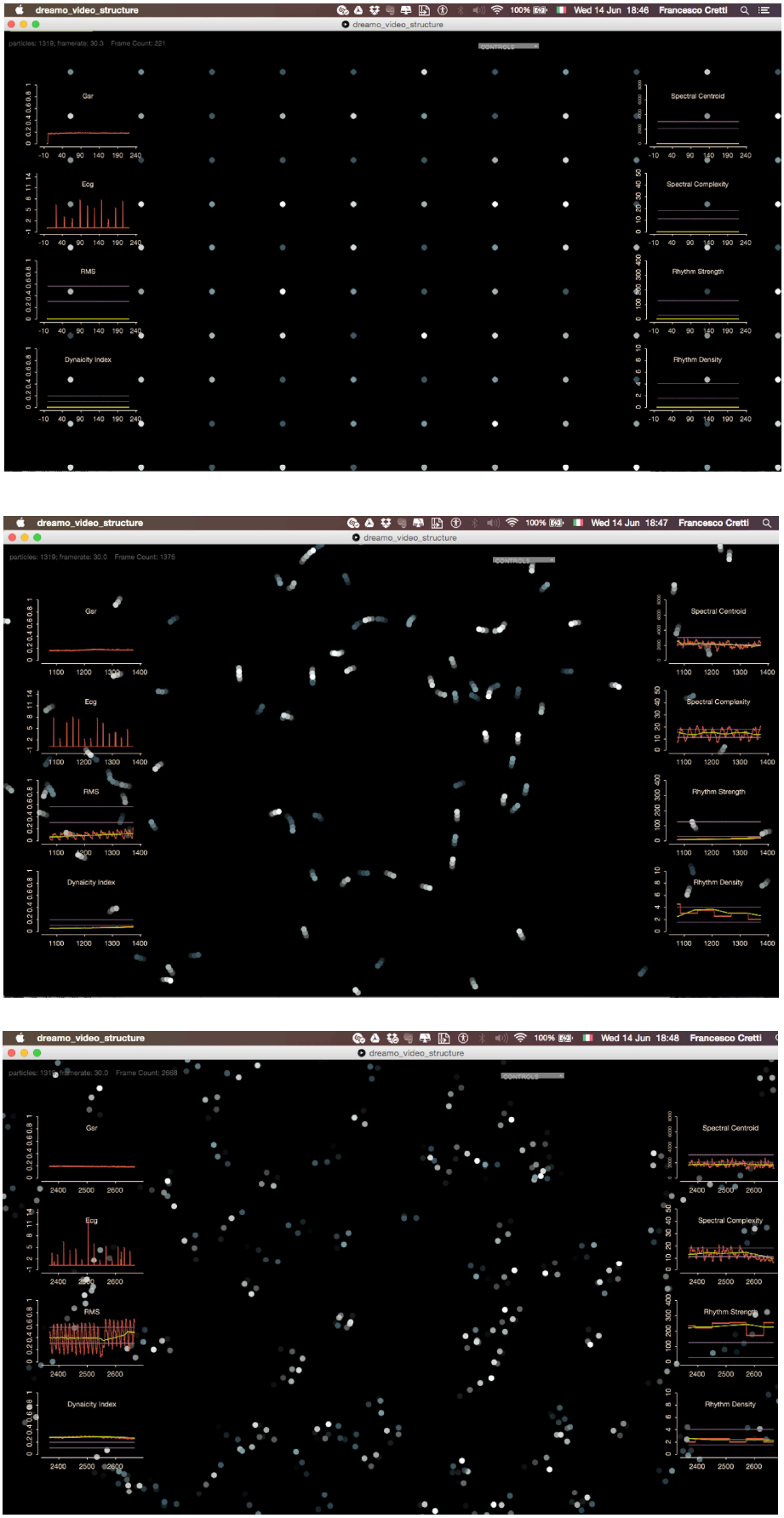
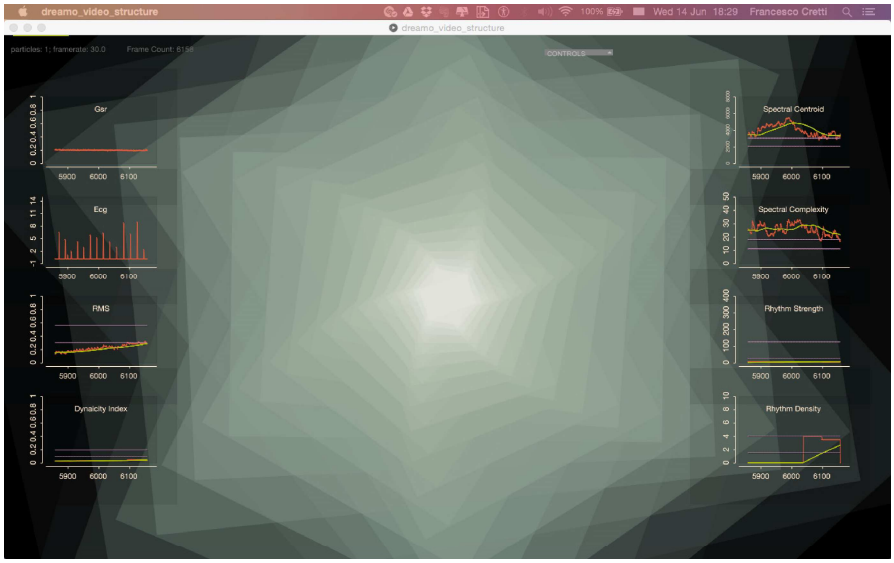


Fig. 5.2: Screenshots tratti dal primo esempio di interazione audio-video in Dreamo

Come si nota dal grafico dell'andamento in tempo reale dell'RMS (il secondo dal basso nella colonna di sinistra), più alto è il valore più le particelle sono veloci nei loro movimenti (e quindi appaiono più separate e disordinate). L'aspetto interessante di questo test non è tanto l'effetto grafico in sé (in questo caso solamente un esempio), quanto il fatto che l'utilizzo del volume RMS per il controllo diretto risulta determinante per trasmettere una sensazione di sincronizzazione audio/ video.

- **Esempio 2: effetto del volume sulla rotazione e della centroide spettrale sulla forma.** Nel secondo test è stato scelto di utilizzare due parametri audio per il controllo di altrettanti parametri grafici. La base grafica di partenza è costituita da una serie di poligoni sovrapposti concentrici (in questo caso 50 rettangoli), ruotati di 45° gradi l'uno rispetto all'altro, con dimensioni e trasparenza decrescenti. Il volume RMS controlla in questo caso il valore della rotazione dell'intera figura, e la centroide spettrale controlla il rapporto fra i due lati dei rettangoli.



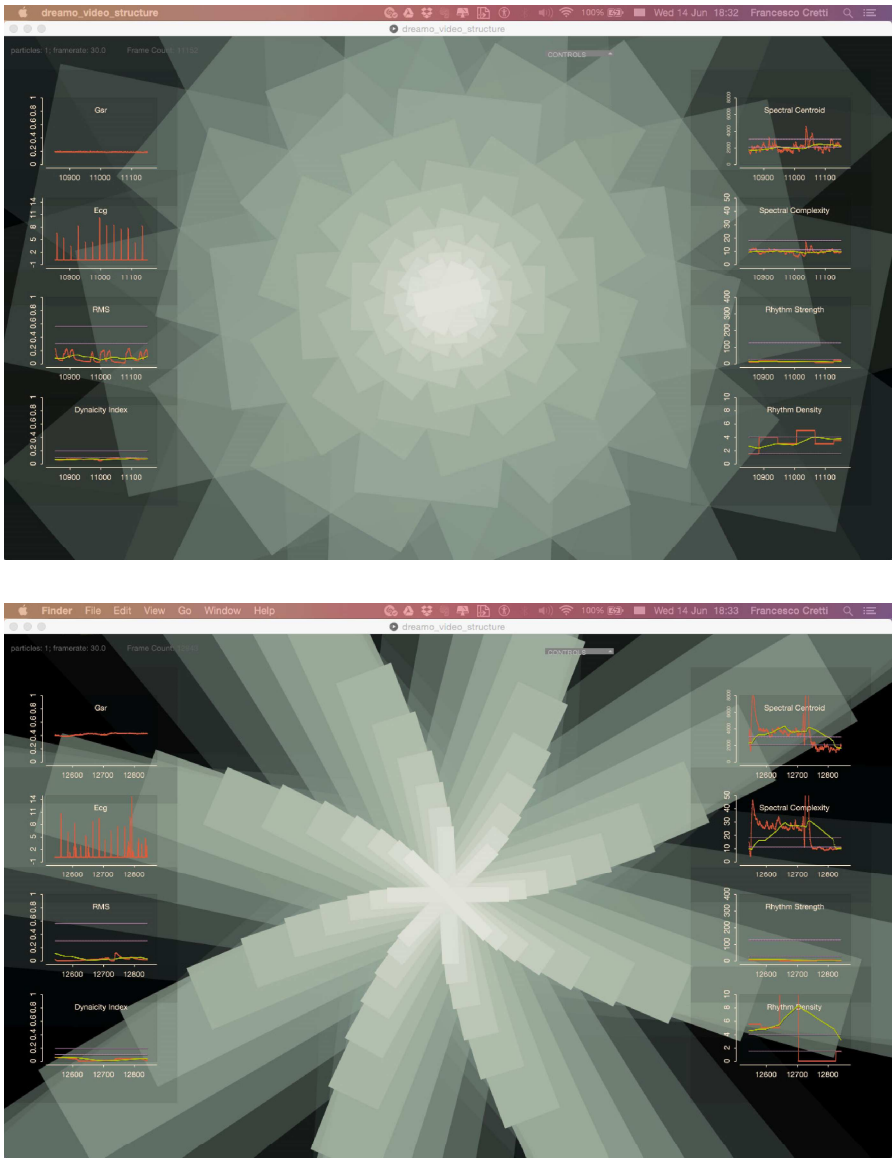


Fig. 5.3: Screenshots tratti dal secondo effetto grafico implementato in Dreamo

Come si nota degli screenshot si genera un effetto "vortice". Questo secondo test ha confermato che l'RMS risulta un parametro fondamentale soprattutto se allacciato ad aspetti che riguardano la fisica degli oggetti in scena (forme, movimento, …) ma ha anche mostrato come le features timbriche possano essere utili, in quanto rappresentano aspetti percettivamente immediati.

- **Esempio 3: effetto del volume su numero di particelle visibili** Il terzo test puntava solamente a cercare una diversa declinazione dell'utilizzo del volume RMS che fosse meno immediata. È stata quindi disegnata una grafica costituita da numerosi punti invisibili collegati fra loro con delle linee (i punti formano la scritta "Dreamo"). L'RMS controlla in questo caso la quantità di collegamenti che

si creano, e di conseguenza la porzione di disegno visibile sullo schermo.

Fig. 5.4: Screenshots tratti dal terzo esempio di scena grafica

Ne risulta che, pur essendo un effetto gradevole dal punto di vista estetico, è meno immediato dal punto di vista del collegamento audio-video. È stato quindi concluso che sia più sensato impostare interazioni di questo tipo con meccanismi di più alto livello, come illustrato nei paragrafi successivi.

5.2.2 Utilizzo dello stato della traccia

Dopo aver sperimentato l'effetto delle assegnazioni dirette, si è voluto sfruttare la rappresentazione dello stato per prendere delle decisioni ad alto livello sul comportamento grafico. Osservando in che stato (alto/medio/basso) si trovano i parametri, e monitorando i loro cambiamenti, è possibile ricavare informazioni utili per innescare reazioni nel video.
Gli stati vengono sfruttati in due modi:

- **parametri aggregati**: sommano il valore dello stato di due o più parametri audio

- **rilevamento dei cambiamenti**: quante features cambiano in un intervallo di tempo

In questo modo è possibile avere un controllo più preciso del comportamento della grafica, effettuato attraverso l'estrazione di alcuni parametri grafici.

Viene ad esempio calcolato un parametro di "elasticità", sfruttando lo stato di parametri quali *dynamicity index* e *rhythm strength*, che può essere utilizzato per influenzare i movimenti degli oggetti in scena.

Oppure viene estratto un indicatore di "vibrazione", che rappresenta l'agitazione ritmica del brano, e che può essere utilizzato per influenzare diversi comportamenti video, a seconda dello scopo del creatore.

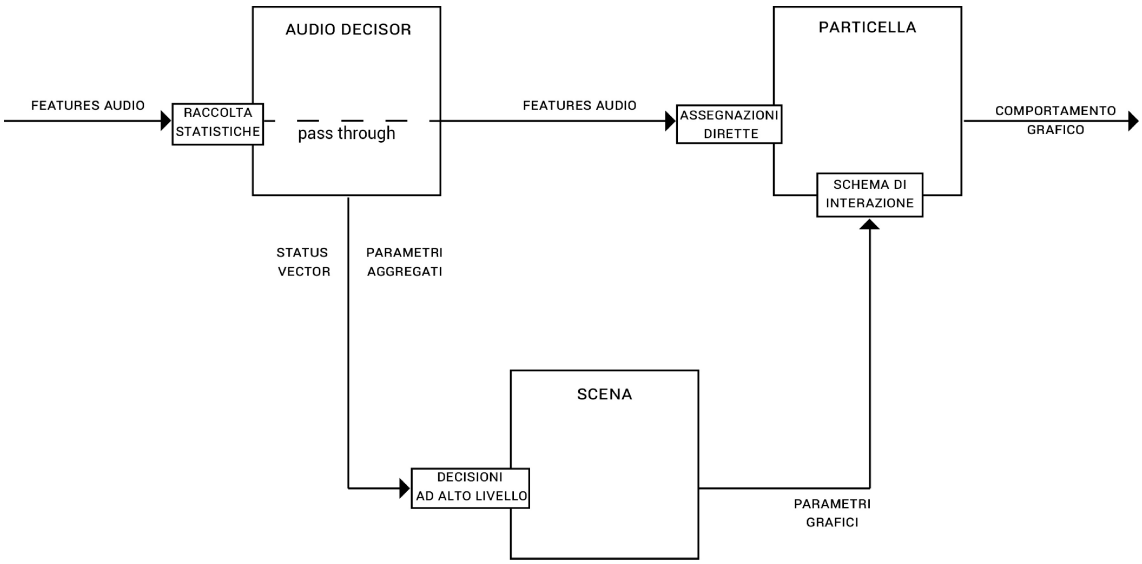


Fig.5.5: Schema a blocchi dei meccanismi di interazione tra audio e video

Tutti questi elementi vengono trattati approfonditamente nel paragrafo successivo, nel quale è descritto il processo di creazione di una grafica animata *mood oriented* che sfrutta questo tipo di interazioni.
Le funzioni che ricavano i parametri grafici sono state implementate della classe Scene, in modo da essere disponibili per tutte le classi figlie. Il codice di tali funzioni è disponibile nell'Appendice C.
In Figura 5.5 è illustrata schematicamente la struttura delle interazioni fra audio e video.

5.3 Creazione di una grafica di esempio

Lo scopo di Dreamo è quello di creare effetti grafici semi-automatici, basati sull'andamento dell'audio in ingresso. Sulla base dei ragionamenti esposti nei paragrafi precedenti, sono stati implementati dei meccanismi di interazione orientati alla rappresentazione del contenuto emotivo della musica, portati poi a compimento in un esempio grafico.
Di seguito sarà quindi illustrato questo processo in tutti i suoi passaggi.
Come esposto nel paragrafo 2.5.1 la struttura di generazione video si basa su un sistema scene/particelle: è stato quindi necessario creare una classe particella, chiamata ***CrazyLines***, che gestisce il tracciamento delle forme e le interazioni dirette con l'audio, ed una scena chiamata ***CrazyL***, che gestisce i controlli a livello più alto.
Nella sezione Appendice D è presente il codice dei files *class_particle_CrazyLines.pde* e *class_scene_CrazyL.pde*.

5.3.1 Forma di base

Per realizzare questo scopo, era necessario partire da una grafica semplice che fosse però in grado di assumere una grande varietà di aspetti diversi. In questo modo sarebbe stato possibile modulare il risultato grafico ottenendo scenari variegati, più interessanti e coinvolgenti agli occhi dello spettatore.

Come già citato, un punto di riferimento importante per la generazione di grafiche con Processing è stato il libro "Generative Design" [24]. In particolare uno dei tanti

esempi presentati in questo volume, sfrutta le funzioni *beginShape()* ed *endShape()*[13] per la creazione di forme complesse. Questo tipo di struttura grafica in Processing si rivela molto versatile e consente di modulare molti parametri ottenendo una grande varietà di forme, ed è stata sfruttata per la programmazione di questo esempio.
La forma di base (in assenza di musica in ingresso) è una semplice linea.

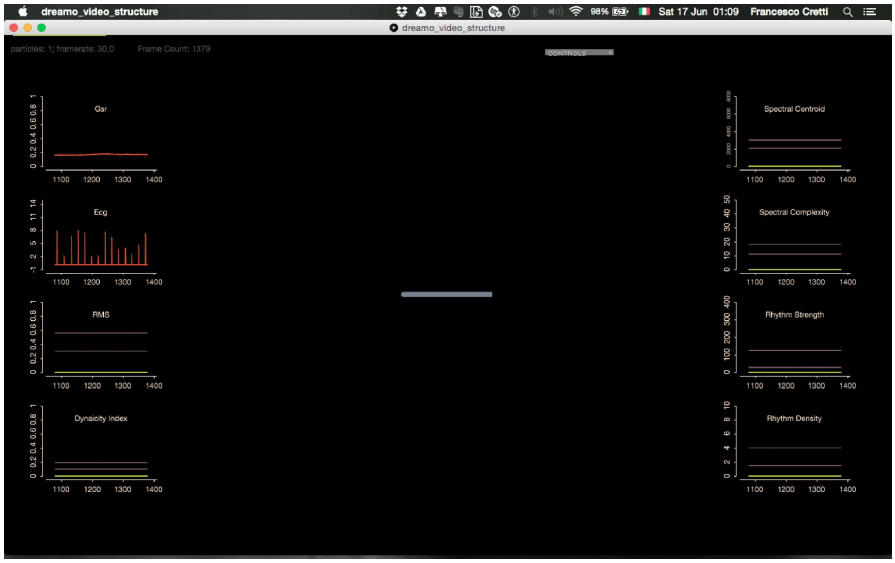


Fig. 5.6: Forma di base dell'effetto grafico

Dopodiché sono state implementate diverse interazioni che consentono di modulare tale forma in base alle caratteristiche acustiche dell'input.

5.3.2 Gestione del colore

Il primo passaggio importante era quello della scelta dei colori da utilizzare nello scenario.
Si è in questo privilegiato l'utilizzo delle features timbriche nella scelta del colore, lasciando agli altri parametri diversi tipi di scelte, come ad esempio quelle sul movimento o sulla frequenza di vibrazione, in modo da ottenere comportamenti più indipendenti e vari.
Era necessario quindi, a partire dalle features timbriche, progettare un meccanismo di scelta del colore.

13 Maggiori informazioni su https://processing.org/reference/beginShape_.html

Questa scelta ha richiesto la revisione di letteratura sulle teorie psicologiche che studiano l'uso o la percezione del colore. Questo ambito di ricerca si è rivelato complesso, ed è stato esplorato con diversi approcci [49–51].

Una prima implementazione è stata fatta sulla base della distinzione tra colori caldi e colori freddi: lo spettro rosso-giallo forma l'insieme dei colori caldi, mentre lo spettro verde-blu rappresenta l'insieme dei colori freddi ([49–51].

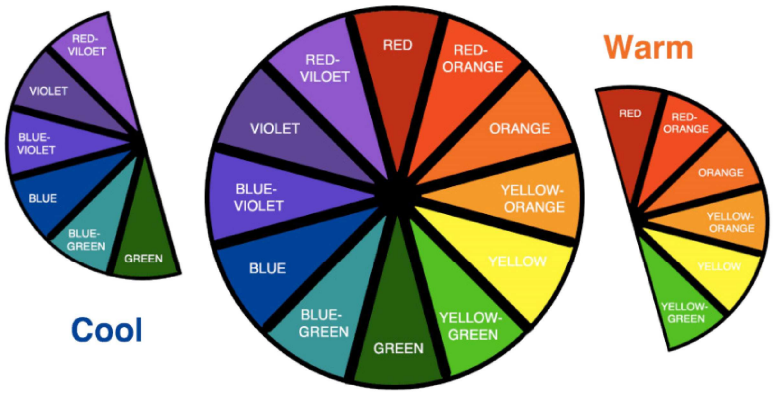


Fig. 5.7: Distinzione tra colori caldi e freddi

Sono stati quindi creati due set di palettes: un set di palettes composte da colori caldi, ed uno di palettes composte da colori freddi. Ogni palette comprende 5 colori diversi.

In Figura 5.8 due esempi di palettes calde e fredde implementate in Dreamo:

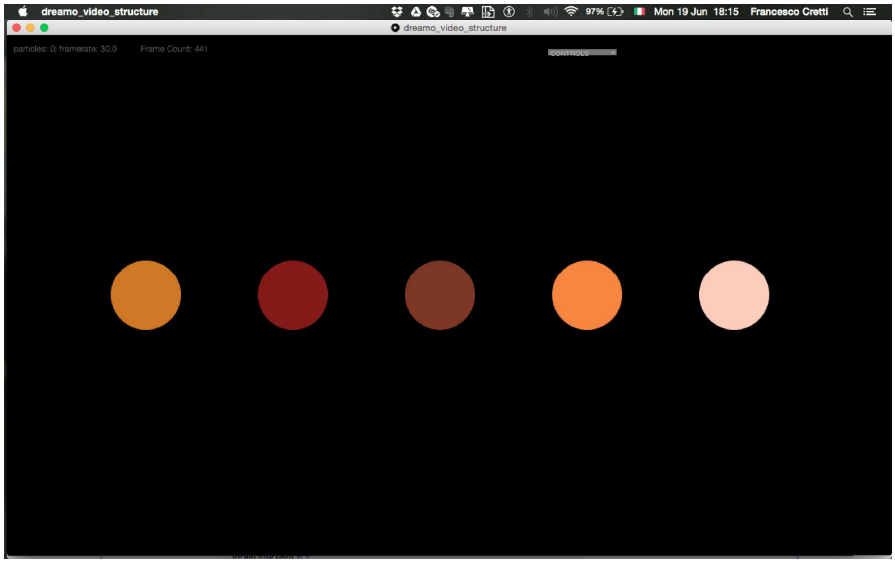


Fig. 5.8: Esempio di palette colori caldi in Dreamo

Fig. 5.9: Esempio di palette colori freddi in Dreamo

In generale la separazione in colori caldi e freddi non è sufficiente per distinguere fra sensazioni positive/negative, ma solo fra sensazioni più intense e più rilassate, come ad esempio sostiene D'Andrade quando afferma che il colore "rosso" è associato al termine "*exciting*" (eccitante), il "giallo" con "*playful*" (gioioso), il "verde" con "*tender*" (tenero) e il "blu" con "*sad*" (triste) e "*solemn*" (solenne) [49–51]).

Sappiamo che valori di centroide e complessità più alte sono legate ad emozioni di tensione [18,42]. Ma anche l'RMS risulta fondamentale nel distinguere queste sensazioni e, poiché si è scelto di non utilizzare le features ritmiche a questo scopo, la preferenza dei parametri da monitorare è ricaduta su:

- RMS
- centroide spettrale
- complessità spettrale

La scelta di mappare i colori utilizzando le informazioni timbriche e l'RMS è presente anche nel lavoro di Soraghan et al. sulla rappresentazione del timbro [52], il quale utilizza tuttavia lo *spectral slope* e lo *spectral spread* al posto della centroide e della complessità spettrali.

Per raggiungere questo scopo abbiamo quindi creato uno score aggregato, composto dalla somma degli stati dei tre parametri scelti (RMS, centroide, complessità), chiamato **paletteIndicator**. In generale è possibile affermare che più alto è tale punteggio, più è alta la sensazione di tensione trasmessa.

Monitorando lo stato dello score aggregato il sistema sceglie se utilizzare una palette

di colori caldi o una di colori freddi in base ad un comparatore di soglia. Quando il punteggio supera la soglia viene scelta una palette di colori caldi, e viceversa. In questo caso la soglia è stata impostata uguale 3, dopo diverse prove ed esperimenti di funzionamento.
Questo meccanismo è racchiuso nella seguente riga di codice, presente nel metodo *update()* della classe **scene_CrazyL.**

```
colorFadeTo(new Palette(choosePaletteFromAudio()),2,audio_decisor.
                        getColorChange());
```

Il metodo ***colorFadeTo()*** permette di ottenere una sfumatura graduale (in questo caso di 2 secondi, come indica il secondo argomento passato alla funzione) da una palette all'altra (il codice di tale funzione è presente nell'Appendice C). Nel momento in cui si passa da una palette all'altra, tutte le particelle presenti nella scena effettuano questa sfumatura. Il terzo argomento della funzione è un valore booleano che si attiva quando lo score aggregato passa attraverso la soglia, innescando quindi il cambiamento del colore.
Dopo diverse prove è stato concluso che questo solo meccanismo non era sufficientemente dinamico e complesso per una gestione del colore adeguata al nostro scopo. Per aumentare il dinamismo dell'effetto grafico era necessario incrementare il numero di reazioni ai cambiamenti dell'audio. Sono state quindi implementate le seguenti funzionalità:

- quando viene rilevato un cambiamento in direzione crescente nello stato della centroide spettrale il sistema aumenta di 60 punti il valore di *brightness* di tutti i colori della palette corrente;

- quando viene rilevato un cambiamento in direzione decrescente nello stato della centroide spettrale il sistema diminuisce di 20 punti il valore di *brightness* di tutti i colori della palette corrente;

- quando viene rilevato un cambiamento in direzione crescente nello stato della complessità spettrale il sistema aumenta di 50 punti il valore di *saturation* di tutti

i colori della palette corrente;

- quando viene rilevato un cambiamento in direzione decrescente nello stato della complessità spettrale il sistema diminuisce di 30 punti il valore di *saturation* di tutti i colori della palette corrente;

```
darkenColors(3,30f, (audio_decisor.getCentroidChangeDir()==-1));
lightenColors(3,50f, (audio_decisor.getCentroidChangeDir()==1));
desaturateColors(3,20f,(audio_decisor.getComplexityChangeDir()==-1));
saturateColors(3,60f,(audio_decisor.getComplexityChangeDir()==1));
```

Inoltre, quando più di 2 features cambiano di stato (in qualsiasi direzione) in un intervallo di tempo di 3 secondi viene innescato un cambio di palette (la scelta fra palette calda e fredda rimane al controllo descritto precedentemente), per avere un feedback visivo immediato rispetto ad un cambiamento presumibilmente importante dell'andamento della traccia audio.

```
colorFadeTo(newPalette(choosePaletteFromAudio()),2,(audio_decisor.
                    getChangesNumber()>2));
```

Infine, alla luce di quanto emerso degli esperimenti sulle assegnazioni diretti di parametri audio a parametri video, è stato deciso di collegare il valore di brillantezza dei colori al valore della centroide spettrale, ed il valore della saturazione a quello della complessità spettrale.

Per consentire questa assegnazione diretta i valori delle due *features* vengono rimappati in un intervallo utile per i rispettivi scopi, ottenendo così un'oscillazione della saturazione (*saturation)* e della brillantezza (*brightness)* intorno ai valori originali del colore compreso nella palette.

Di seguito uno schema che illustra tutti i meccanismi di controllo del colore implementati:

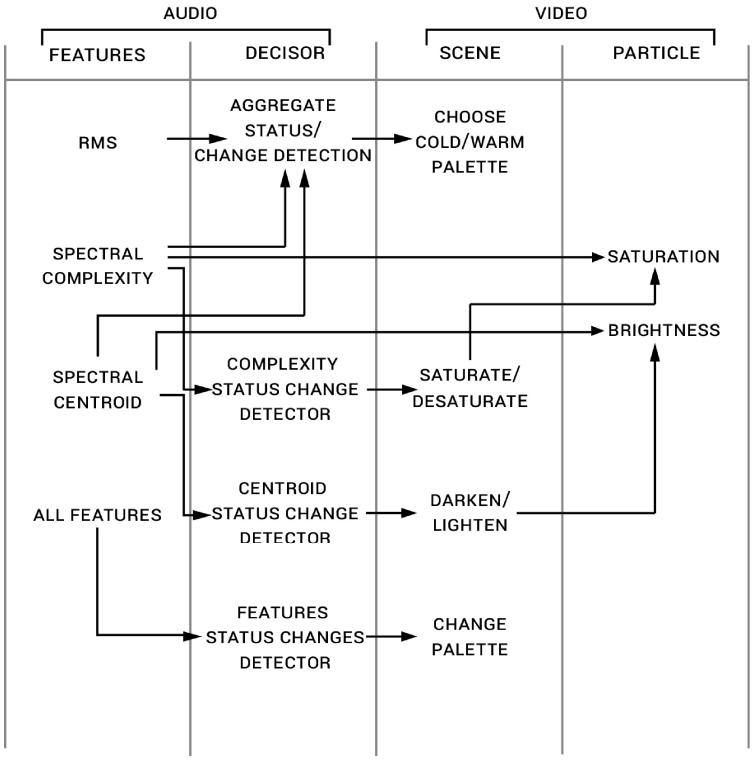


Fig. 5.10: Visualizzazione schematica del controllo del colore

Per concludere possiamo affermare che risulta tuttavia difficile comprendere l'apporto delle varie caratteristiche acustiche sulla percezione del "colore della musica". Inoltre spesso brani musicali vengono immediatamente associati al contesto temporale, sociale e culturale in cui sono stati composti o vengono solitamente suonati, che si porta dietro un immaginario fatto di figure, simboli e, inevitabilmente, colori.

La nostra è una delle possibili proposte, e siamo consapevoli che ce ne possano essere molte altre. Ad esempio, un'alternativa possibile è associare l'intensità alla saturazione, poiché sempre secondo le ricerche di D'Andrade le persone associano la sensazione di "debolezza" con colori chiari e poco saturi [49].

5.3.3 Gestione dei movimenti

Sulla base di quanto esposto nella sezione 5.2.1, è stato deciso di associare il valore del volume RMS a parametri che influenzassero il movimento delle particelle in scena. Rimaneva quindi importante capire quali fossero i più adatti ed efficaci. Dopo diverse prove la scelta è ricaduta su due movimenti: pulsazione o rotazione. Con pulsazione si intende la variazione della scala (ottenuta grazie alla funzione *scale()* di Processing), mentre cone rotazione si intende la variazione della posizione angolare (grazie al metodo *rotate()* di Processing). Entrambi vengono appunto controllati direttamente all'RMS ed il sistema sceglie se innescare un movimento rotatorio od una pulsazione.
La pulsazione si presta bene a rappresentare canzoni con un ritmo molto presente e marcato, avendo una corrispondenza immediata tra due misure di grandezza (volume RMS e dimensione della figura) che risulta di naturale associazione. A supporto di questo, le animazioni di Oskar Fischinger sono state un importantissimo punto di partenza. [31,32]

La rotazione di forme geometriche ha invece un effetto più rilassante ed ipnotico, ed è adatta nel caso di brano con un ritmo debole o addirittura assente. [53,54]

Per consentire quindi al sistema di capire quale movimento scegliere (o quale combinazione dei due), è stato creato un parametro aggregato, **elasticityIndicator**, definito come la somma dei valori di stato di due features importanti per il rilevamento della presenza del ritmo:

- Dynamicity index
- Rhythm strength

Viene quindi generato un "coefficiente di elasticità" che varia da 0.4 a 2.2 e che aumenta con l'aumentare del parametro **elasticityIndicator**, ed un "coefficiente di rotazione" che varia da 0 a 0.1 e diminuisce all'aumentare dell'**elasticityIndicator.**
Tali valori vengono poi passati alla particella, che li utilizza come coefficienti moltiplicatori nella generazione dei propri movimenti.
In questo modo, quando viene rilevato un brano con ritmo forte e marcato viene scelto un movimento elastico di pulsazione mentre quando un brano non presenta un netto

contenuto ritmico viene scelto un movimento rotazionale. Sono chiaramente possibili varie combinazioni dei due movimenti.

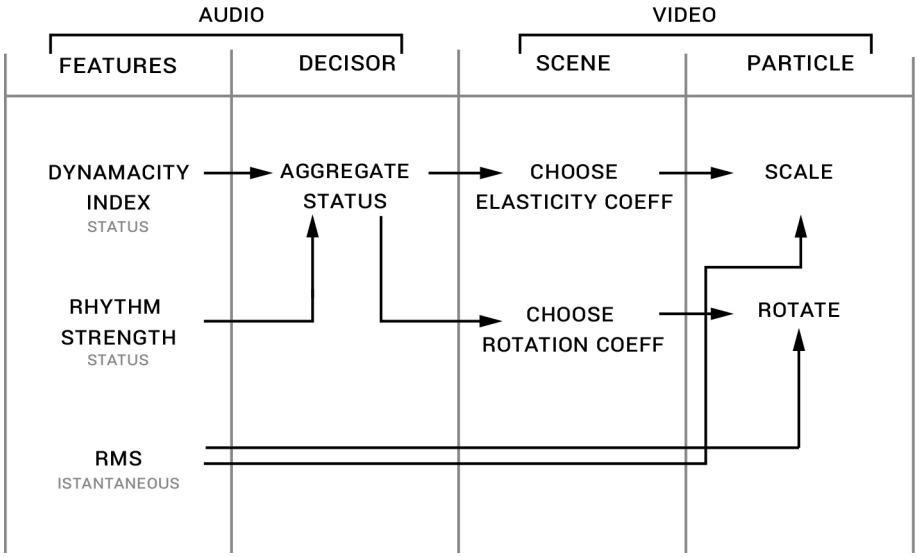


Fig. 5.11: Visualizzazione schematica del controllo movimenti

5.3.4 Gestione delle linee

Un altro aspetto percettivamente interessante è sicuramente lo spessore delle linee.
Da un punto di vista grafico timbri più chiari e aperti trovano un'ottima corrispondenza con tessiture e linee più sottili. In questo senso un'opera di notevole impatto è la Rheo: 5 horizons di Kurokawa [37], che associa suoni rumorosi (quindi ad alto contenuto frequenziale) a texture composte da linee sottili.
Il controllo sul modo in cui vengono tracciate le linee è uno strumento espressivo molto importante e permette di ottenere figure molto diverse dal punto di vista della percezione emotiva.
Sono state implementate delle funzioni che permettono la gestione delle linee attraverso tre diversi parametri:

- **Spessore di base,** ***(weightSeed)***

- **Range di vibrazione,** ***(vibrationRane)***

- **Frequenza di vibrazione, *(vibrationFreq)***

Lo spessore varia ad ogni frame: viene infatti scelto uno spessore random che dipende dal valore dello spessore di base e del range di vibrazione. Questo valore è detto spessore target (*target weight*) e deve essere raggiunto dopo un numero finito di iterazioni.

```
targetWeight=random(weightSeed-vibrationRange/2,weightSeed+vibrationRange/2);
```

Ad ogni frame lo spessore viene incrementato o decrementato (a seconda se lo spessore target è maggiore o minore dello spessore corrente) di una quantità che corrisponde al valore della frequenza di vibrazione. Quando lo spessore raggiunge (o supera) il target, viene generato un nuovo target random, con la relazione sopra descritta.

Con questo meccanismo è possibile quindi controllare lo spessore delle linee e la sua variazione: con il valore *weightSeed* si controlla lo spessore di base, il parametro *vibrationRange* decide entro quale distanza dallo spessore di base possono essere assegnati i *targetWeight*, mentre la vibrationFreq controlla il tempo con cui lo spessore raggiunge il target e di conseguenza la frequenza della vibrazione.

Le funzione *changeStrokeWeight()* è quella che si occupa di aggiornare il valore che viene poi passato al metodo *strokeWeight()* nativa di Processing, che di fatto determina lo spessore della linea che viene visualizzata. Il codice di tale funzione è visibile nel file *class_CrazyParticle.pde* (Appendice D).

Era a questo punto necessario stabilire come i tre parametri grafici venissero controllati dall'audio.

Lo spessore viene scelto con una relazione inversamente proporzionale al timbro: un parametro aggregato detto **timbreIndicator** viene calcolato attraverso la somma dei valori di stato delle due features timbriche implementate in Dreamo:

- Centroide spettrale

- Complessità spettrale

Viene in seguito generato un valore che varia da 40, quando il **timbreIndicator** è minimo, a 3 quando è massimo. Tale valore è assegnato al parametro grafico

weightSeed. In questo modo timbri più chiari e complessi corrispondono a linee sottili mentre timbri più scuri e smooth trovano una buona corrispondenza percettiva in linee più spesse.

La frequenza di vibrazione ha un riscontro visivo importante: una frequenza più veloce trasmette un senso di agitazione, movimento ed energia. Per questo motivo è stato deciso di lasciare questa decisione ad un parametro aggregato chiamato **vibrationIndicator**, composto dalla somma dei valori di stato di:

- Rhythm Density
- RMS

Successivamente una funzione sceglie un valore che varia da 1 (quando **vibrationIndicator** è minimo) a 3.8 (quando **vibrationIndicator** è massimo), il quale viene assegnato al parametro **vibrationFreq** della particella.
In questo modo le linee di grafiche corrispondenti a brani con alte densità di onsets e valori di energia elevati oscilleranno in maniera molto rapida, risultando percettivamente coerenti. D'altro canto brani meno intensi dal punto di vista ritmico presenteranno una vibrazione più lenta, che meglio si addice a quel tipo di sonorità.

Il range di vibrazione viene invece scelto in maniera inversamente proporzionale al volume RMS: in questo modo si ottengono dei risultati visivi convincenti, in base anche ai valori degli altri parametri grafici, le linee presentano dei comportamenti che vanno da una sorta di effetto "respirazione" ad un effetto quasi "stroboscopico"; il parametro grafico *vibrationRange* viene calcolato con la seguente relazione:

$$vibrationRange = \frac{3}{RMS}$$

In questo caso è stato scelto di utilizzare l'RMS medio, calcolato su un intervallo di tempo di 2 secondi.
È stato inoltre aggiunto un controllo per evitare la tendenza ad infinito del valore di *vibrationRange* quando valori di RMS sono molto bassi: quando l'RMS medio è

inferiore a 0.05 il valore in uscita viene fissato a 3/0.05=60.
Ricapitolando, il controllo dei tre parametri grafici per lo spessore delle linee:

- *wightSeed* (spessore di base): cresce all'aumentare del **timbreIndicator**;
- *vibrationFreq:* cresce all'aumentare del **vibrationIndicator**.
- *vibrationRange*: inversamente proporzionale al volume RMS;

Combinazioni diverse di questi tre parametri creano una notevole varietà di forme e comportamenti. Anche in questo caso si tratta solamente una proposta di soluzione su tante possibili, che tuttavia, almeno per quella che è l'esperienza personale di chi scrive, trova un immediato riscontro dal punto di vista percettivo ed emotivo.

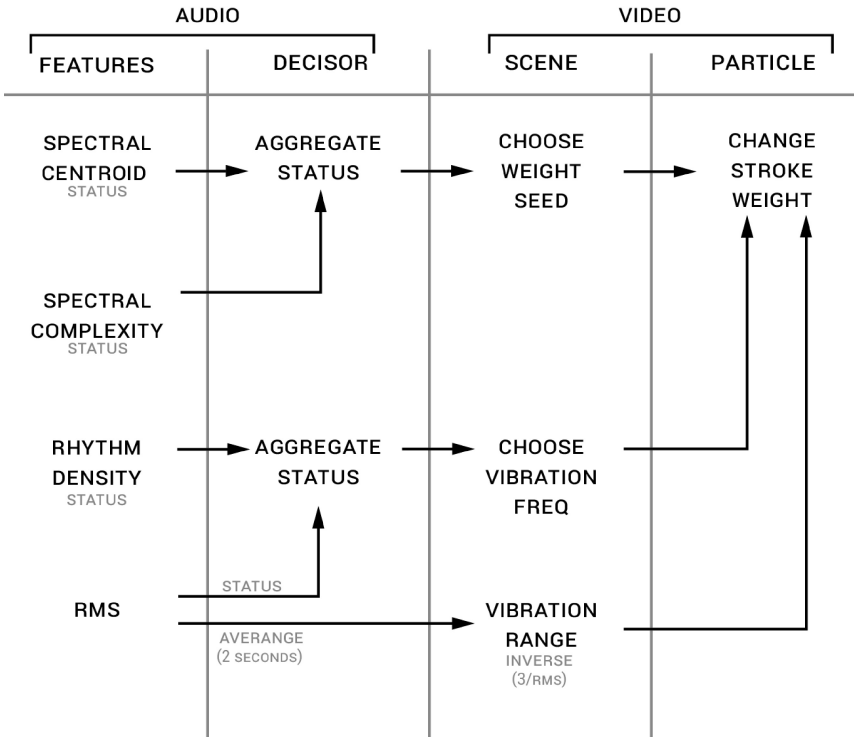


Fig. 5.12: Visualizzazione schematica del controllo delle linee

5.3.5 Gestione della forma/distorsione

Grazie alla struttura *beginShape() / endShape()* di Processing, è possibile controllare la posizione in cui vengono generati i vertici della figura geometrica, tramite il metodo *curveVertex(x,y)*, il quale aggiunge un vertice in posizione (x,y). Questo consente di distorcere o deformare la figura semplicemente cambiando la posizione di generazione dei suoi vertici.
È stato scelto di vincolare la posizione dei vertici esterni della figura alla centroide spettrale: è stato utilizzato un parametro che moltiplica le coordinate dei vertici esterni e che dipende in modo esponenziale dal valore della media della centroide spettrale, calcolata su un intervallo di 2 secondi precedenti.

$$par1 = 5^{\frac{spectral\ centroid}{2500}}$$

In questo modo alte frequenze portano i vertici esterni più lontani dal centro, espandendo così le linee e trasmettendo una sensazione di tensione, facendo addirittura in alcuni casi uscire le linee dallo schermo.

L'RMS (la sua media calcolata su 2 secondi) controlla invece i vertici al centro della forma: volumi più elevati aumentano il discostamento tra i punti trasmettendo quindi un'idea di distorsione. La risoluzione della forma viene invece controllata dalla complessità spettrale: più il brano è complesso armonicamente, più la forma risulterà complessa (più vertici vengono generati).

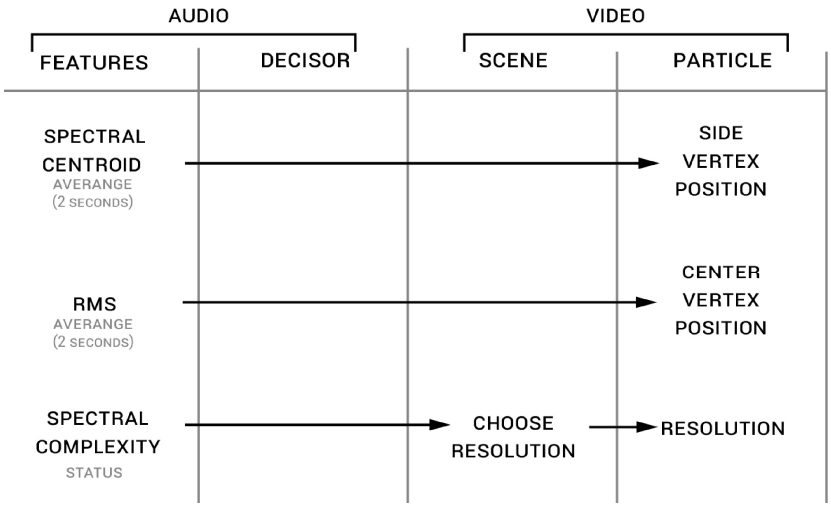


Fig. 5.13: Visualizzazione schematica del controllo della forma

5.3.6 Esempio di funzionamento

Di seguito saranno riportati i risultati visivi di alcuni esperimenti fatti dando in input all'algoritmo appena descritto alcuni brani musicali, caratterizzati da stile, sonorità, genere e mood diversi fra loro.

Le due tabelle seguenti riassumono tutti i meccanismi d'interazione implementati, dalle assegnazioni dirette alle decisioni basate sugli stati descritti nel paragrafo precedente.:

ASSEGNAZIONI DIRETTE

PARAMETRO AUDIO	PARAMETRO GRAFICO	RELAZIONE
RMS istantaneo	Scala (pulsazione)	Lineare (diretta)
RMS (media 2 secondi)	Rotazione	Lineare (diretta)
RMS (media 2 secondi)	Distorsione (posizione dei vertici centrali)	Lineare (diretta)
Centroide spettrale (media 2 secondi)	Distorsione (posizione vertici esterni)	Esponenziale
Centroide spettrale (media 2 secondi)	Brillantezza colori	Lineare (diretta)
Complessità spettrale (media 2 secondi)	Saturazione colori	Lineare (diretta)
RMS (media 2 secondi)	Vibration Range	Lineare (inversa)

Tabella 5.1: Assegnazioni dirette di features acustiche a parametri grafici

DECISIONI IN BASE ALLO STATO

PARAMETRO AUDIO	PARAMETRO GRAFICO	TIPO DI SCELTA	RELAZIONE
Centroide spett.(stato) Complessità spett. (stato) RMS (stato) [*paletteIndicator*]	Colore	Scelta palette (calda/ fredda)	Soglia (booleano)
Complessità (stato)	Saturazione	aumento/diminuzione saturazione di un valore fissato	Cambiamento di stato
Centroide (stato)	Brillantezza	aumento/diminuzione brillantezza di un valore fissato	Cambiamento di stato
Tutte le features (stato)	Colore	Cambio palette	Cambiamento di stato
Dyn. Index (stato) Rhythm Strength (stato) [*elasticityIndicator*]	Movimento (pulsazione)	Controllo del coefficiente che moltiplica l'RMS in *scale()*	Direttamente proporzionale
Dyn. Index (stato) Rythm Strength (stato) [*elasticityIndicator*]	Movimento (rotazione)	Controllo del coefficiente che moltiplica l'RMS in *rotate()*	Inversamente proporzionale
Complessità spett. (stato) Centroide spett. (stato) [*timbreIndicator*]	Spessore	Controllo del valore di *weightSeed*	inversamente proporzionale
Rhythm Density (stato) RMS (stato) [*vibrationIndicator*]	Frequenza di vibrazione	Controllo del valore di *vibrationFreq*	direttamente proporzionale
Complessità spettrale (stato)	Complessità della forma grafica	Controllo del valore di *formResolution*	direttamente proporzionale

Tabella 5.2: Decisioni di reazioni grafiche prese in base allo stato della traccia

Si nota che ogni controllo dipende da un set differente di variabili. È possibile quindi ottenere una varietà ampia di forme.

Di seguito riportati alcuni screenshots di risultati grafici ottenuti dai test su canzoni.

MASSIVE ATTACK - ANGEL

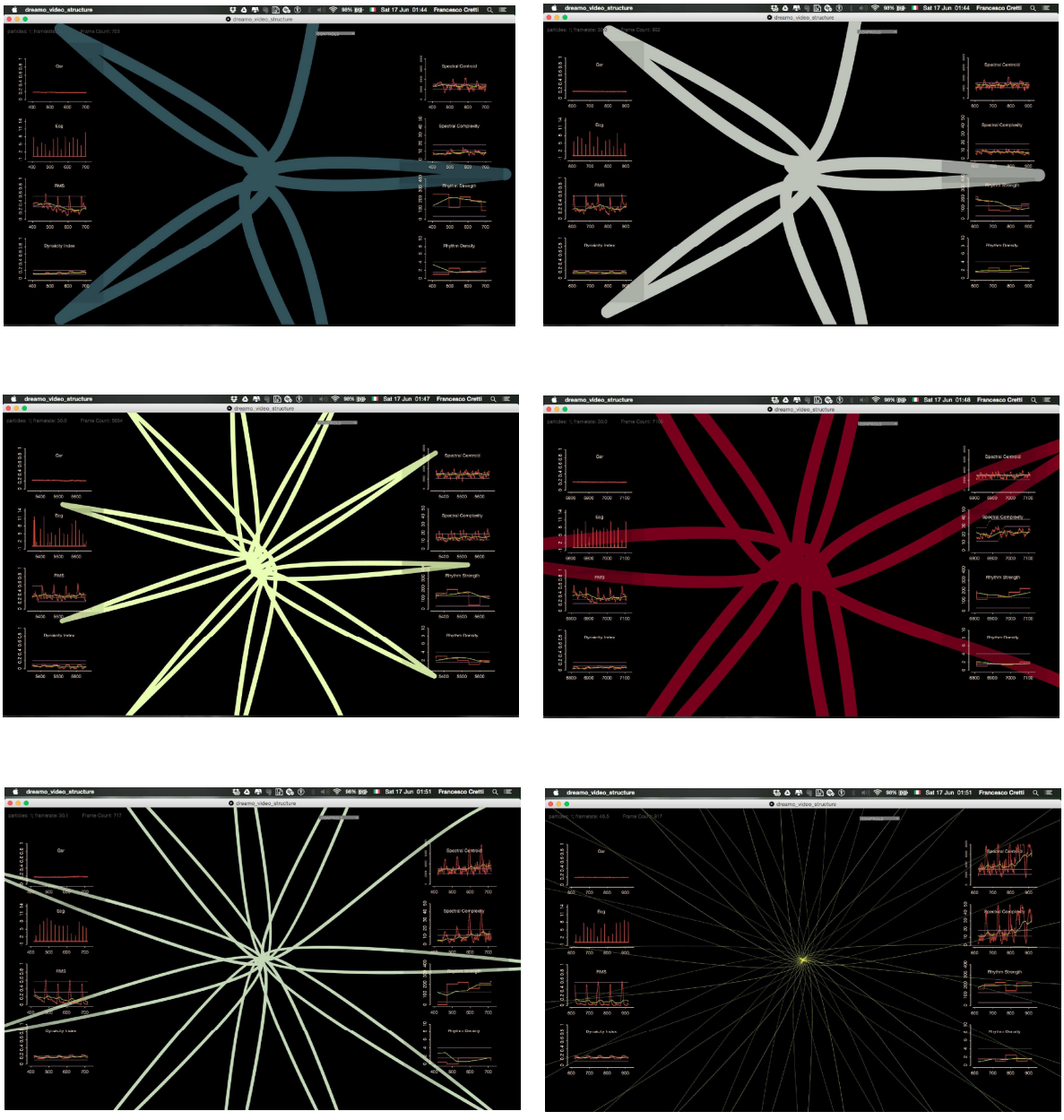


Fig. 5.14: Grafiche generate dal sistema durante varie fasi del brano "Massive Attack - Angel"

FAT FREDDY'S DROP - SHIVERMAN

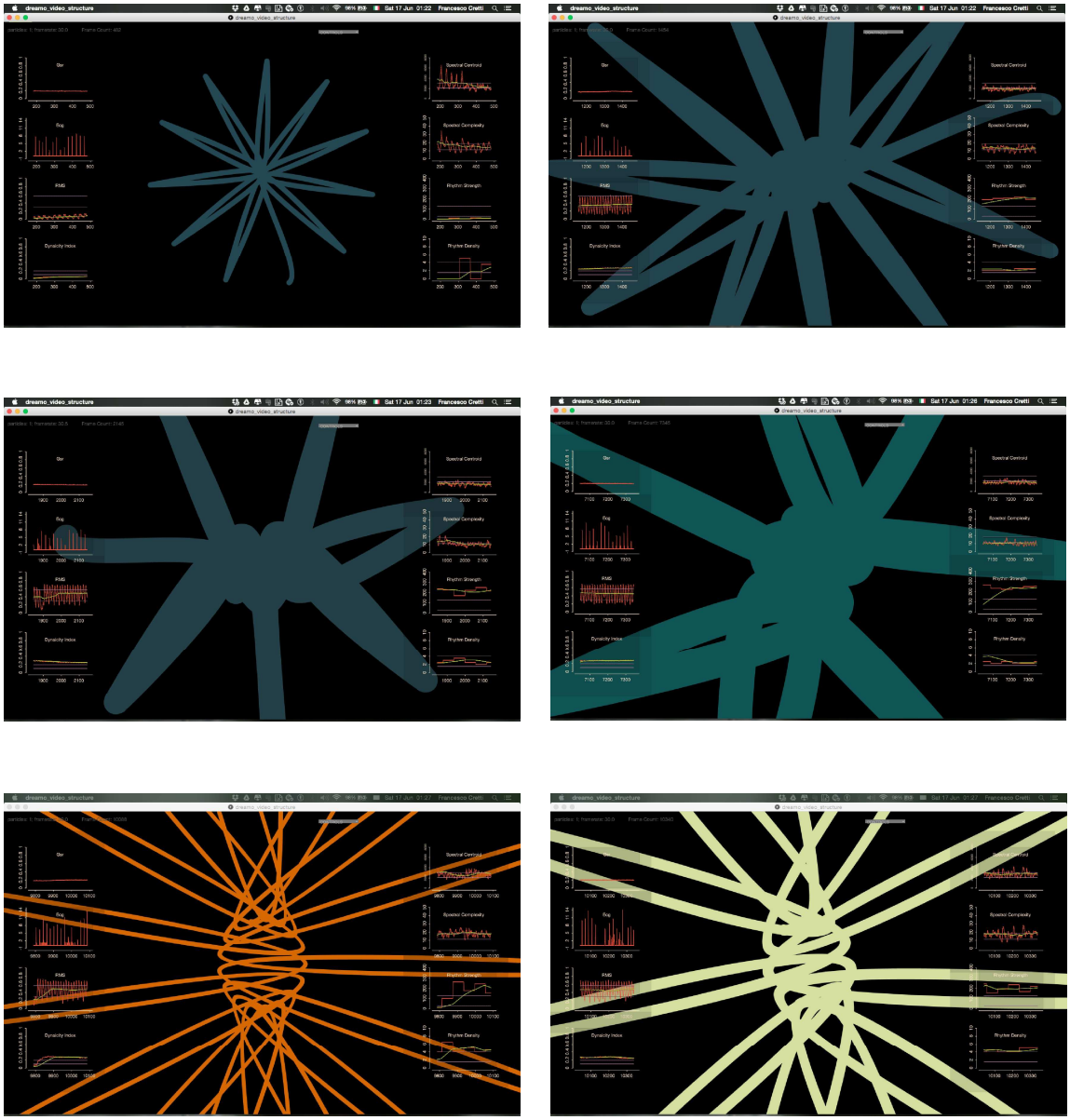


Fig. 5.15: Grafiche generate dal sistema durante varie fasi del brano "Fat Freddy's Drop - Shiverman"

NATHAN FAKE - THE SKY WAS PINK

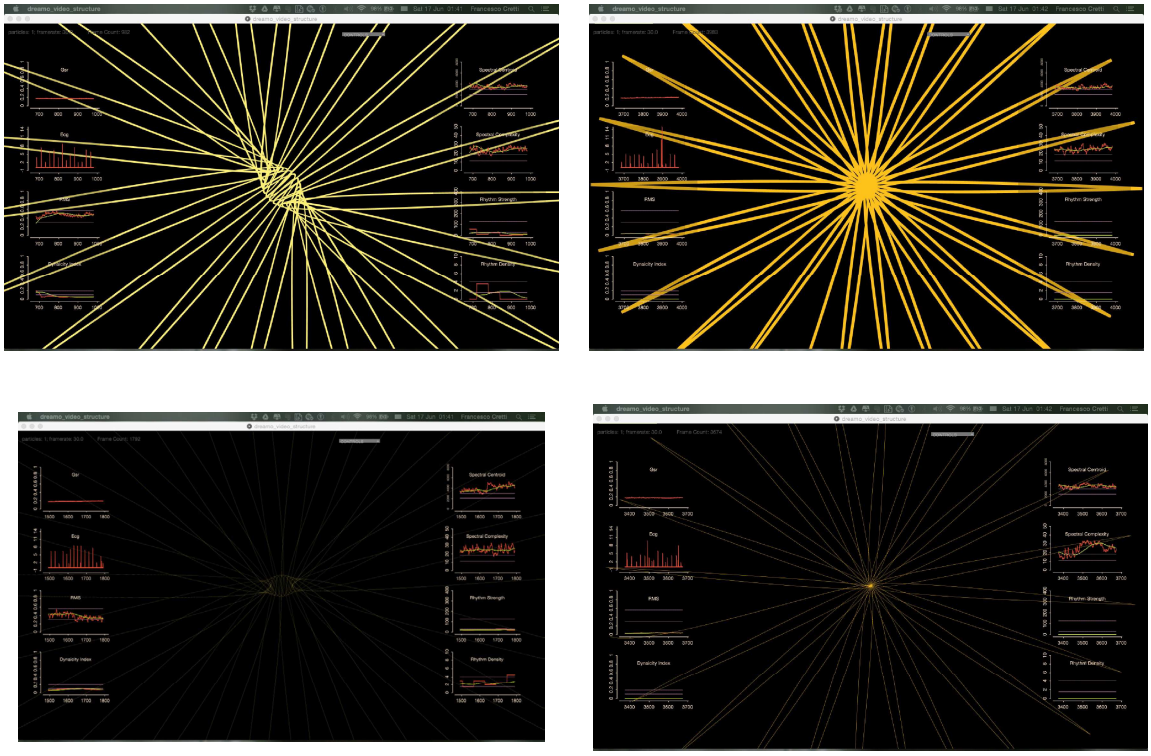


Fig. 5.16: Grafiche generate dal sistema durante varie fasi del brano "Nathan Fake - The Sky Was Pink"

TOMMY GUERRERO - THANK YOU MK

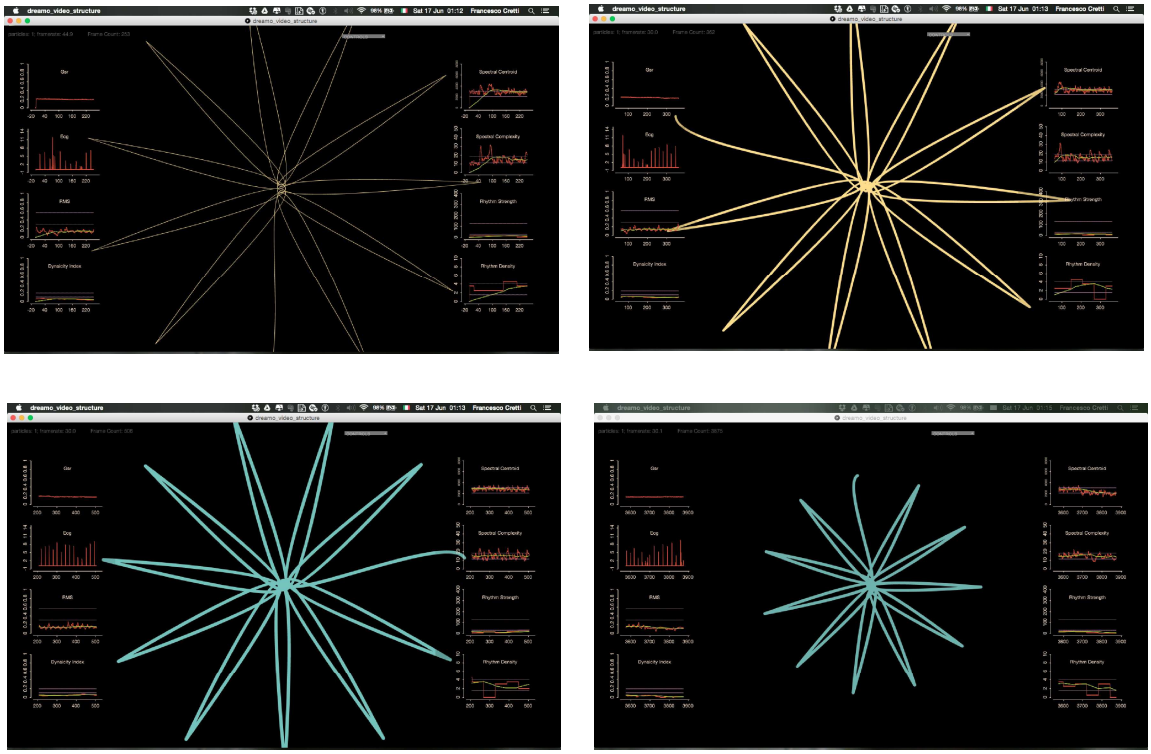


Fig. 5.17: Grafiche generate dal sistema durante varie fasi del brano "Tommy Guerrero - Thank You MK"

RED SNAPPER - GET SOME SLEEP TIGER

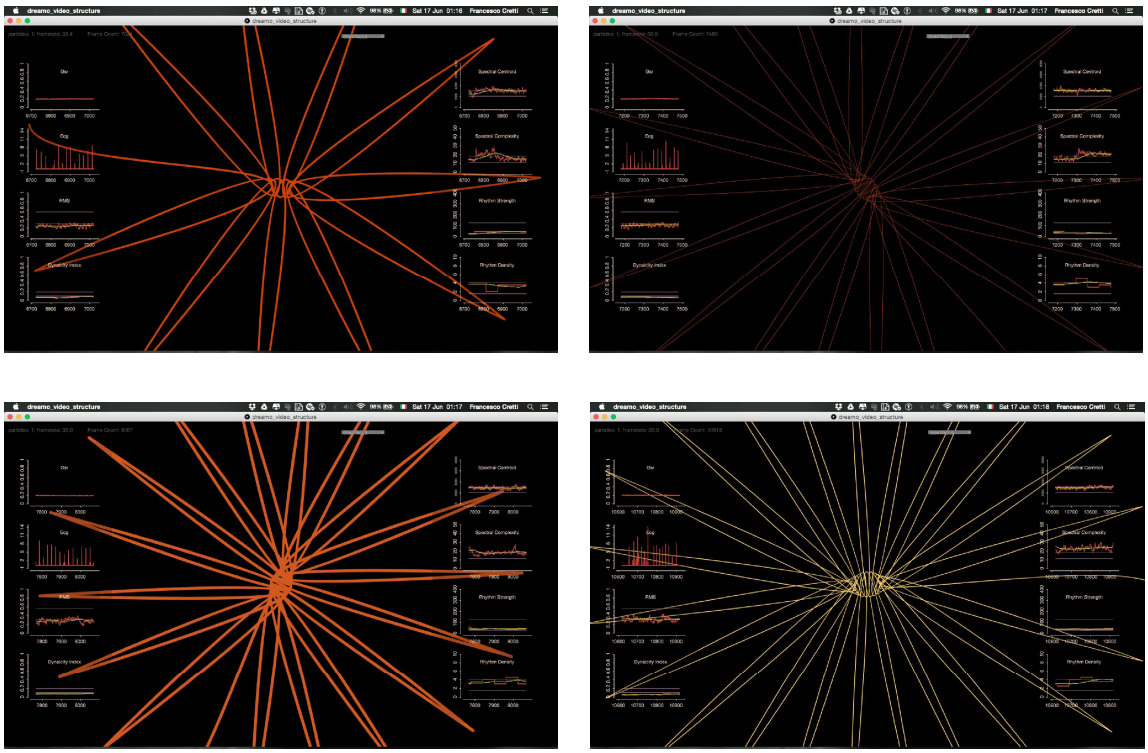


Fig. 5.18: Grafiche generate dal sistema durante varie fasi del brano "red snapper - get some sleep tiger"

ALVA NOTO & RYUICHI SAKAMOTO - MOON

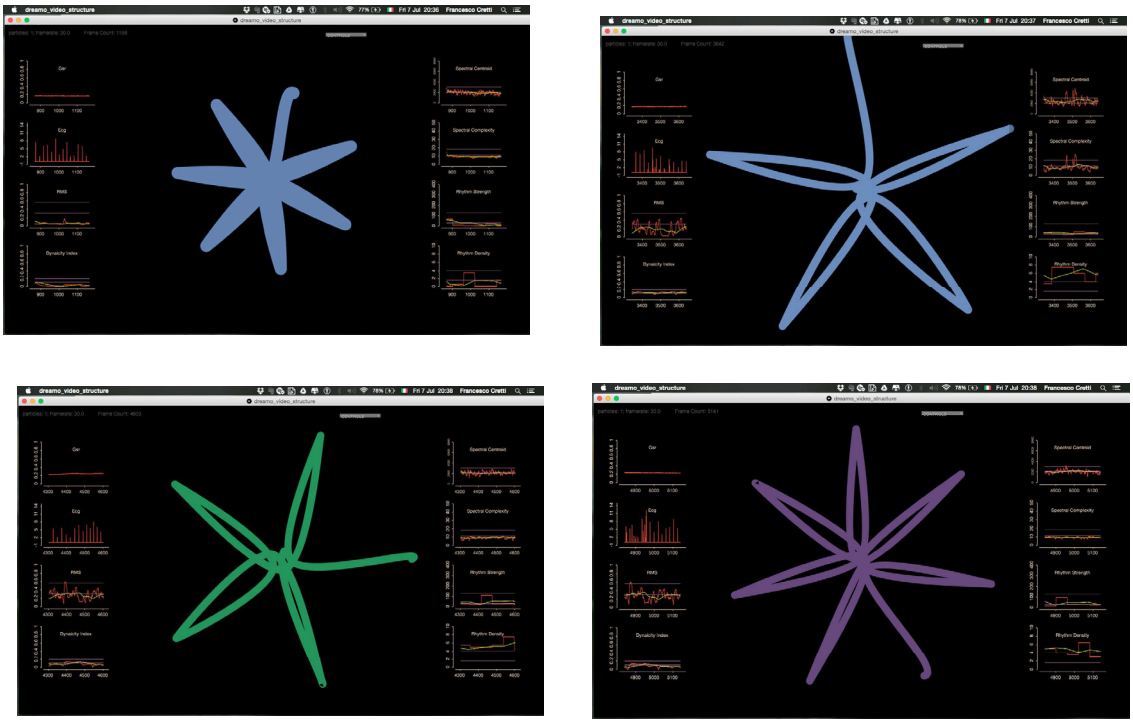


Fig. 5.19: Grafiche generate dal sistema durante varie fasi del brano "Alva Noto & Ryuichi Sakamoto - Moon

5.4 Dreamo come strumento: tutorial di utilizzo

Dreamo si prefigge di essere strumento a disposizione per chiunque voglia cimentarsi nella creazione di grafiche responsive all'audio.

L'insieme dei meccanismi implementati per la demo sopra descritta, sono un esempio di quello che è possibile generare con il nostro software, ed è stato progettato grazie all'ispirazione di lavori artistici presenti in letteratura, ma anche in base al gusto personale di chi scrive.

È chiaramente possibile creare le più disparate forme grafiche che, a partire dalle stesse informazioni estratte dal motore di analisi audio, si comportino in modo diverso, a seconda delle intenzioni di chi le implementa.

Come altri software (ad esempio Max/MSP), Dreamo vuole lasciare all'utente finale la possibilità di scegliere a proprio piacimento, ed in base alle proprie esigenze, cosa viene generato dal motore grafico.

Di seguito sarà quindi illustrato brevemente il processo per la creazione di un effetto grafico con regole di interazione diverse da quello presentato come demo in questa tesi, creato con un'idea diversa di realizzazione.

5.4.1 Passaggio 1: creazione di una particella

Come ripetutamente affermato la particella è l'elemento grafico di base, stabilisce le regole di disegno, e costituisce il cuore dell'effetto grafico.
Per creare una particella è quindi necessario creare una nuova classe, che sia ereditata dalla classe generica Particle.

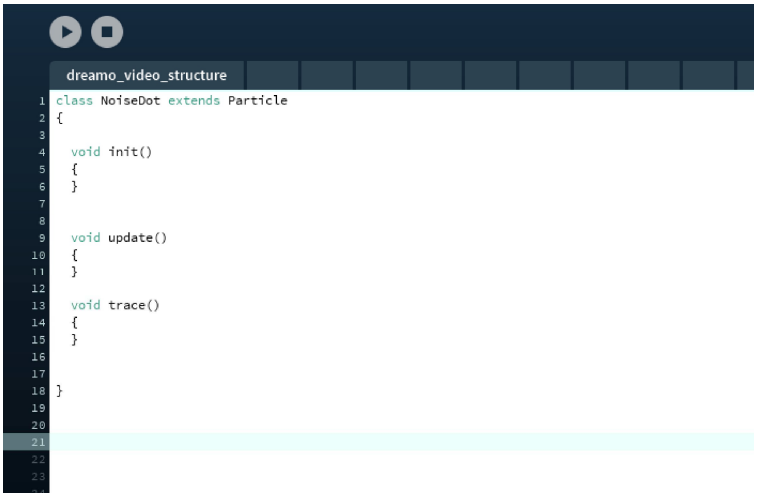


Fig. 5.20: Creazione della classe particella

A questo punto è necessario stabilire le regole di disegno nella funzione *trace(),* quelle di aggiornamento nella funzione *update()* e le impostazioni iniziali nella funzione *init()*. Per questo esempio si è pensato alla particella come un circoletto che cambia colore. Per la generazione del circoletto si è utilizzata la funzione *ellipse()* di Processing.

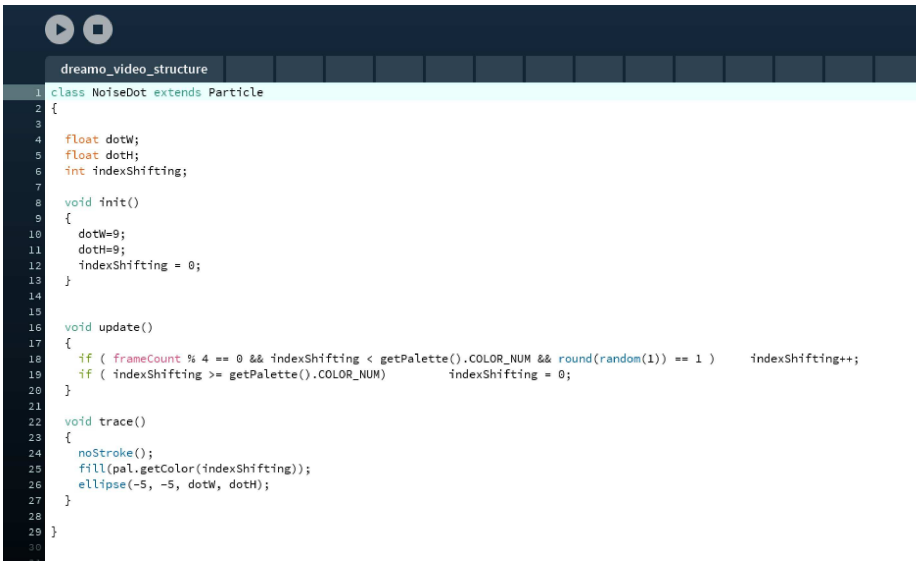


Fig.5.21: Implementazione dei metodi init(), update()e trace() della particella

5.4.2 Passaggio 2: creazione di una scena

Il secondo passaggio è quello di realizzare una classe scena, che controlla il comportamento delle particelle.

Fig. 5.22: Creazione di una classe scena

Nella definizione della scena non è necessario sovrascrivere il metodo *trace()*, perché il modo in cui la scena viene tracciata è definito solamente dalla particelle che essa contiene, ma solamente i metodi *init()* ed *update()*.

Per creare una griglia di particelle:

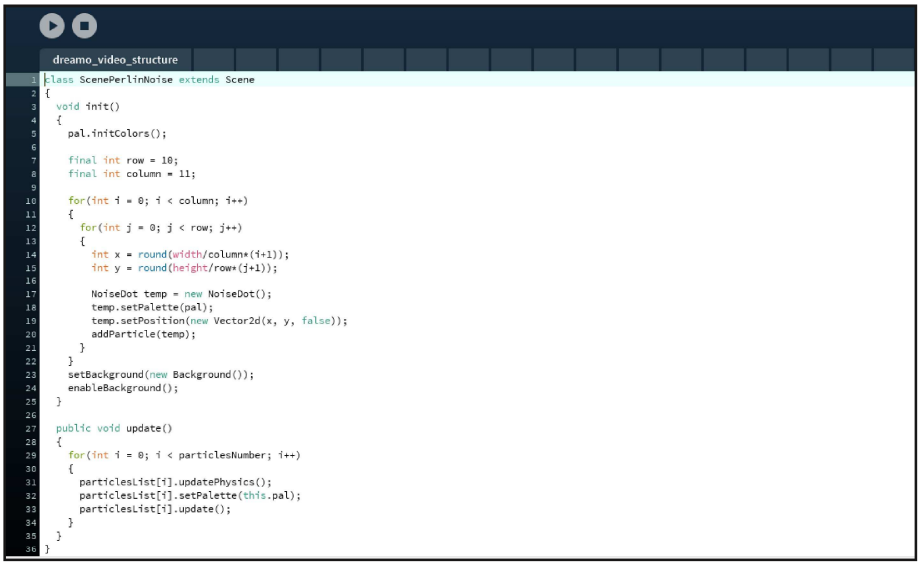


Fig. 5.23: Implementazione dei metodi init e update della scena

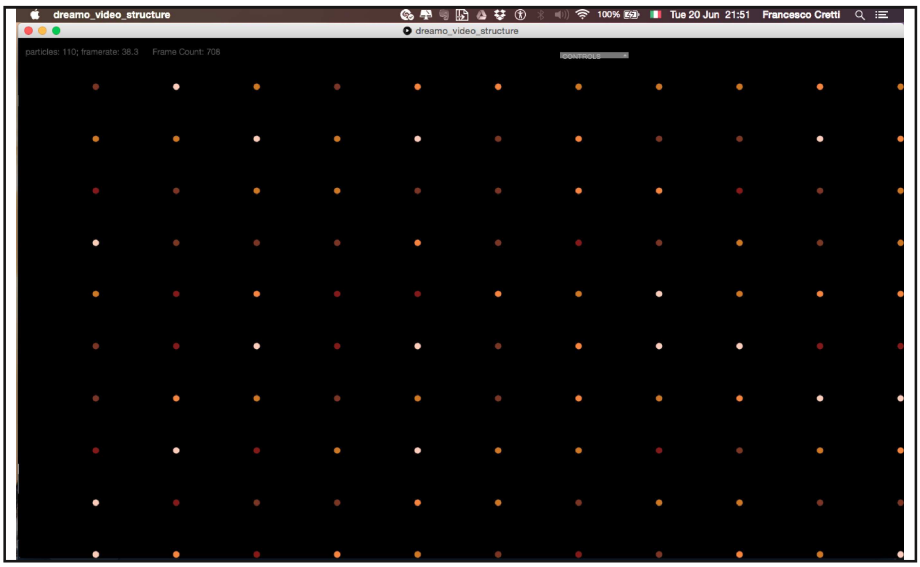


Fig. 5.24: Screenshot del risultato grafico

In generale, le operazioni fondamentali per poter visualizzare una particella si trovano nel metodo init() della classe scena e sono le seguenti:

```
myParticle p = new myParticle(); //crea una nuova particella
p.setPalette(pal); //passo la palette della scena corrente
p.setPosition(new Vector2s(x,y,false)); //scelgo la posizione nella scena
addParticle(p); //aggiungo la particella alla scena
```

5.4.3 Passaggio 3: impostazione dei meccanismi di interazione

ASSEGNAZIONI DIRETTE

A questo punto, dopo aver creato una grafica di base, è necessario scegliere ed impostare i meccanismi di interazione. È consigliato stabilire le assegnazioni dirette nel codice della particella mentre le decisioni ad alto livello nel codice della scena.

L'obiettivo di questo esempio è che le particelle "danzino" seguendo la musica: si vuole che il sistema prenda alcune decisioni qualitative in base allo stato dell'audio, influendo così su alcune caratteristiche del movimento. Inoltre questa volta la saturazione sarà controllata direttamente dal volume RMS, e non dalla complessità spettrale come nell'esempio precedente.

Per cominciare bisogna stabilire come si crea il movimento, agendo su due proprietà di

ogni particella:

- Modulo della velocità;
- Direzione della velocità;

Per avere un'immediata sincronizzazione dei movimenti con l'andamento dell'audio, è sufficiente assegnare il valore del volume RMS al modulo della velocità.
La direzione della velocità viene invece stabilita in modo randomico: per avere un effetto più vario e naturale alla vista, si è scelto di utilizzare il rumore di Perlin [55] al posto della classica funzione *random()*. Il rumore di Perlin è un algoritmo per la creazione di sequenze numeriche pseudo-random, e trova applicazioni in moltissimi campi, ogniqualvòlta si necessiti di un andamento casuale apparentemente più "naturale" e realistico.
Ogni volta che viene richiamata la funzione *update()* della particella (quindi ad ogni frame) viene generato un nuovo valore dell'angolo che stabilisce la direzione della traiettoria. Questa la relazione:

$$angle = noise\left(\frac{posX}{noiseScale}, \frac{posY}{noiseScale}\right) * noiseStrength$$

Il metodo *noise()* restituisce un valore del rumore di Perlin, *noiseScale* è un parametro che controlla le coordinate in ingresso al metodo, mentre *noiseStrength* è un parametro di scala che ne moltiplica il valore in uscita, che è sempre compreso fra 0 e1.
In generale, più è alto il valore di *noiseScale* più bassa è la distanza fra le coordinate in ingresso al metodo, e più smooth risulta la sequenza dei valori in uscita. Allo stesso modo, bassi valori del parametro *noiseScale* si traducono in sequenze di numeri più turbolente.

È possibile affermare che *noiseScale* controlla la frequenza, mentre *noiseStrength* controlla l'ampiezza della successione numerica restituita dalla funzione.
Nel seguente screenshot sono visibili le linee di codice necessarie a mettere in atto le interazioni fin qui descritte:

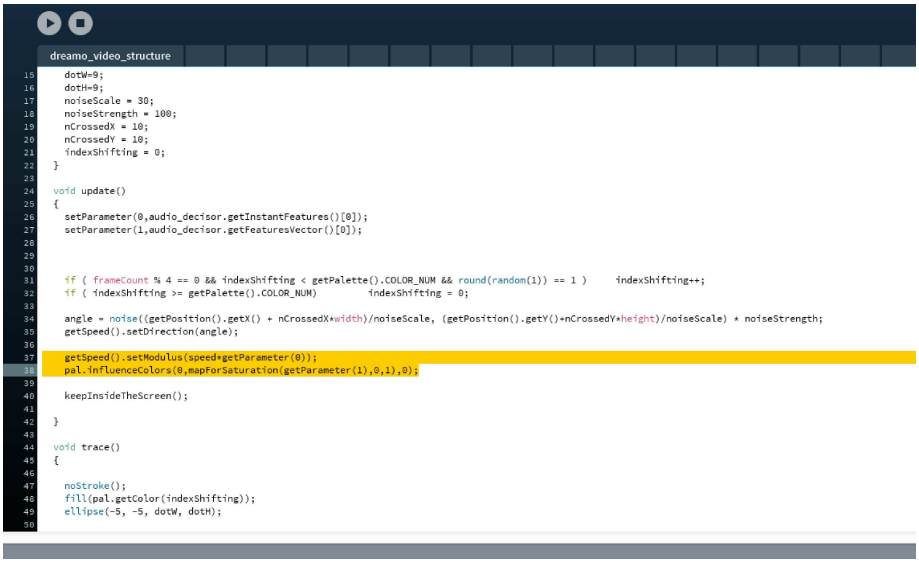


Fig. 5.25: Implementazione del movimento e delle assegnazioni dirette

Le linee di codice sottolineate sono quelle che mettono in atto l'assegnazione diretta dell'RMS alla posizione delle particelle e alla saturazione del colore (notare che la funzione ***influenceColor***s richiede 3 argomenti: hue, saturation, brightness: quando viene passato 0 come argomento il valore corrispondente rimane fisso al valore originale).

Per queste assegnazioni viene sfruttato un array di parametri general purpose implementati nella classe generale Particle e quindi ereditato da tutte le particelle. Assegnando a questi parametri un valore desiderato tramite il metodo *setParmeter(N, value)*, è possibile recuperare tale valore in qualsiasi punto del codice con il metodo *getParameter(N)*.

La linea 35 è invece quella che assegna il valore dell'angolo calcolato a seconda del rumore di Perlin alla direzione della velocità della particella.

Nella linea di codice 34 sono presenti altre due variabili (*nCrossedX, nCrossedY*), necessari alla funzione ***keepInsideTheScreen()***, che mantiene le particelle sempre all'interno dello schermo visibile.

DECISIONI AD ALTO LIVELLO

Il secondo livello d'interazione sul quale si può agire sono le decisioni ad alto livello, ovvero quelle che il sistema prende in base allo stato dei parametri audio.

Come descritto precedentemente, sono stati creati i seguenti metodi implementati nella classe **Scene**, che dipendono da set di features diverse, e sono quindi utili per controllare questo tipo

di meccanismo:

- *choosePaletteFromAudio() -> RMS / centroid /compleixity*
- *chooseElasticityFromAudio() -> DynIndex / Rhythm Strength*
- *chooseRotationFromAudio() -> DynIndex / Rhythm Strength (inversa)*
- *chooseThicknessFromAudio() -> Centroide / Complexity (inversa)*
- *chooseVibrationFromAudio() -> RMS / Rhythm Density*
- *chooseResolutionFromAudio() -> Complexity*

In questo contesto si punta a realizzare un semplice esempio al fine di dimostrare le modalità di utilizzo di Dreamo, e quindi le interazioni da implementare sono state limitate a quattro:

1. scelta della palette calda/fredda in base a RMS e alle features timbriche;
2. ad un timbro scuro dovranno corrispondere particelle più grandi, e viceversa;
3. ad una maggiore densità di ritmo dovrà corrispondere una vibrazione delle particelle con frequenza maggiore;
4. ad un ritmo più marcato le particelle dovranno "agitarsi" rimanendo sul posto, mentre per ritmi più leggeri le particelle dovranno effettuare spostamenti più lineari.

L'interazione numero 1 è identica all'esempio precedente. Per attivarla è necessaria una riga di codice, che agisce sulla palette della scena, da inserire nel metodo *update()* della scena stessa:

```
colorFadeTo(new Palette(choosePaletteFromAudio()),2,audio_decisor.
                        getPaletteChange());
```

Per il secondo meccanismo. Possiamo sfruttare la funzione ***chooseThicknessFromAudio()***, la quale ritorna un valore che diminuisce all'aumentare delle features timbriche. Questo valore può essere poi assegnato al diametro delle particelle, ottenendo così il risultato desiderato.

Risulta quindi necessario passare tale valore numerico alle particelle presenti nella scena. Per fare questo sfrutto nuovamente i parametri generici, utilizzando questa volta la funzione *setParameter* nel metodo *update()* della scena. Nell'*update()* della particella è possibile a questo punto recuperare il valore con la funzione *getParameter*, ad assegnarla alle variabili che determinano il diametro del circoletto.

Dopo una serie di test è risultato necessario aggiungere un coefficiente moltiplicativo pari a 1.5.

```
particlesList[i].setParameter(2,1.5*chooseThicknessFromAudio());
```

Per i meccanismi di interazione 2 e 3, era ragionevole agire sui parametri che controllano il rumore di Perlin. Cambiando infatti i coefficienti *noiseScale* e *noiseStrength*, cambia notevolmente la sequenza di angoli pseudo-random che vengono associati alla direzione delle particelle, cambiando così radicalmente il tipo di movimento.
Per l'interazione 3, è possibile sfruttare il metodo *chooseVibrationFromAudio()*, che ritorna un valore rappresentativo dell'agitazione ritmica del brano. A causa di come è stata impostata la funzione che genera gli angoli, era necessario impostare una relazione inversamente proporzionale. Dopo diverse prove è stato inoltre scelto un coefficiente moltiplicatore pari a 1200.

```
particlesList[i].setParameter(3,1200/chooseVibrationFromAudio());
```

Il parametro 3 viene poi assegnato alla variabile *noiseScale*: più è denso il ritmo della canzone più basso sarà il valore della variabile, generando così un movimento più "frenetico" delle particelle.

L'interazione numero 4 sfrutta invece la funzione **chooseElasticityFromAudio()**, che restituisce un valore crescente maggiore è la presenza di un ritmo all'interno della

traccia. Tale valore viene poi assegnato alla variabile *noiseStrength*, che determina l'ampiezza della sequenza numerica del rumore di Perlin, a cui è stato aggiunto un coefficiente moltiplicatore pari a 200. In questo modo in presenza di ritmi molto marcati, *noiseStrength* assume valori molto alti e di conseguenza aumenta il range di generazione dei valori dell'angolo (fino a [0-440°]), rendendo di fatto equiprobabili tutte le direzioni di movimento. Ciò che ne risulta è una vibrazione sul posto. Se invece il ritmo è poco evidente, o addirittura assente, vengono generati angoli all'interno di un ristretto range (il più piccolo è [0-80°]) generando movimenti più lineari.

```
particlesList[i].setParameter(4,200*chooseElasticityFromAudio());
```

Gli screenshots seguenti mostrano le porzioni di codice che mettono in atto la completezza dei meccanismi illustrati precedentemente.

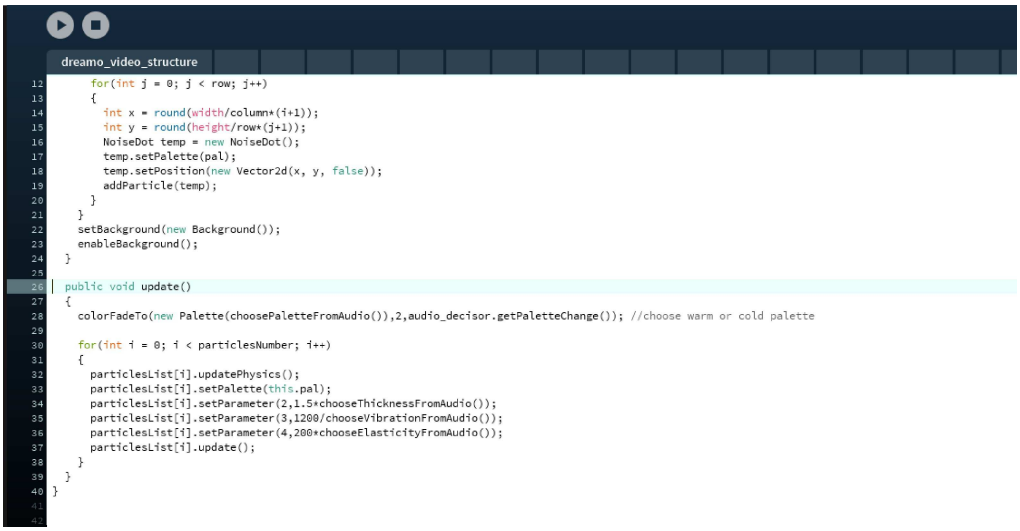


Fig. 5.26: Nel metodo update() della scena vengono assegnati i parametri generici ai parametri calcolati in base allo stato delle features (ed eventualmente adattati attraverso coefficienti moltiplicatori)

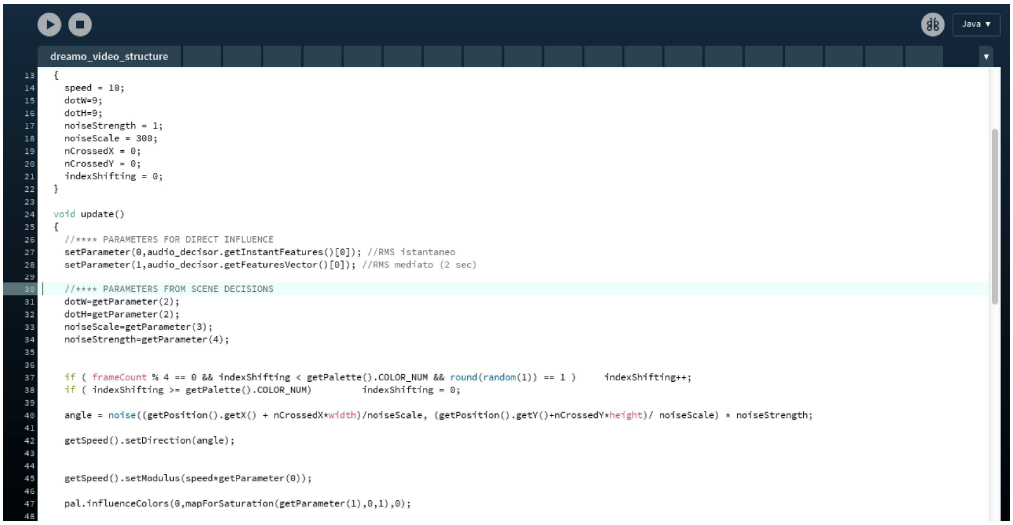


Fig. 5.27: Nel metodo update() della particella vengono effettuate le assegnazioni dei parametri generici alle variabili che influenzano il comportamento grafico della particella stessa

Per riassumere, il seguente schema presente le interazioni audio/video presenti in questo effetto grafico, la cui implementazione è stata descritta nella guida appena conclusa.

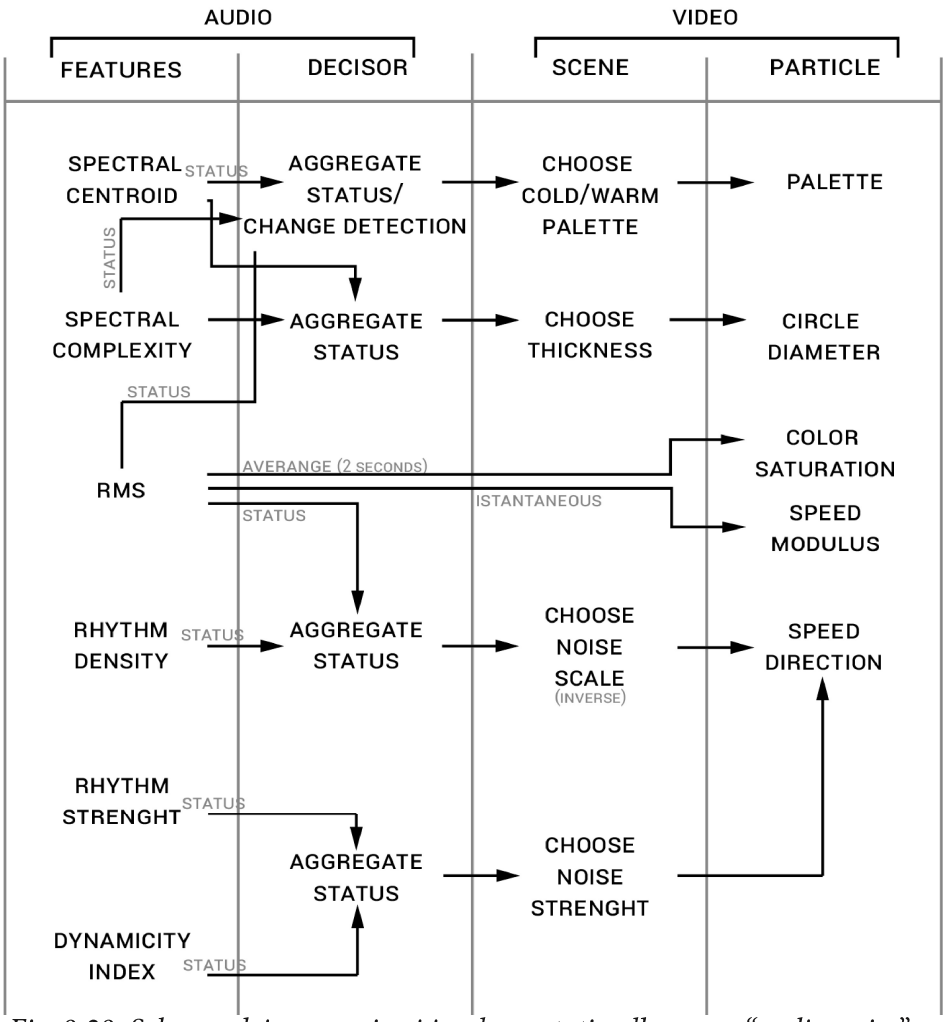


Fig. 8.28: Schema dei meccanismi implementati nella scena "perlin noise"

In conclusione, questo è un breve esempio che dimostra come è possibile creare una grafica influenzata dall'audio, scegliendo il tipo di forme e comportamento. È chiaramente possibile creare una particella più semplice e stabilire le regole di comportamento più disparate, in base allo scopo ed al gusto personale.

In questo caso chi scrive si è occupato sia di creare le grafiche che le funzioni che le influenzano. Questo tool potrebbe ad esempio essere utilizzato da due persone diverse: un primo artista crea la grafica con dei parametri ne determinano i comportamenti, mentre un secondo le "anima" associando tali parametri ai dati estratti dall'audio secondo la propria idea di interazione.

Capitolo 6

Conclusioni

6.1 Dreamo ad oggi

Dreamo è un progetto che nasce prima nella fantasia che sulla carta ed abbraccia numerose materie richiedendo competenze trasversali (audio e signal processing, ingegneria biomedica, machine learning e intelligenza artificiale, elettronica e comunicazioni, microcontrollori, grafica e design).
Questo passaggio dalla teoria alla realizzazione pratica ha comportato non poche difficoltà, soprattutto nella fase iniziale del progetto.
I primi mesi sono stati interamente dedicati alla pianificazione (sempre teorica) dei passi da intraprendere.
Alla costruzione della struttura software necessaria alle operazione di base è stata quasi totalmente dedicata la prima fase operativa del progetto. Mentre la parte hardware è stata testata in un secondo momento non appena ci è pervenuto il materiale.
L'enorme quantità di informazioni disponibili sul web si è dimostrata essere, previa accurata selezione, una risorsa preziosa per il progetto.

A partire dallo stato dell'arte e dagli strumenti proposti in letteratura, è stata sviluppata una soluzione originale e adeguata allo scopo del progetto, che permette di generare effetti grafici in tempo reale rappresentativi delle sensazioni sonore espresse da un brano musicale. La definizione dello "stato" ha consentito di impostare i meccanismi d'interazione non solo a livello delle associazioni dirette, ma anche di capire le evoluzioni che avvengono all'interno del brano e di prendere decisioni a livello più alto. L'analisi di lavori artistici e lo studio di ricerche psicologiche (come ad esempio quelle sul colore) ha supportato la progettazione di comportamenti grafici significativi dal punto di vista emotivo.
Inoltre il lavoro svolto finora rappresenta uno strumento a disposizione di chiunque voglia, magari con un diverso background artistico e culturale, sperimentare soluzioni nell'ambito dell'interazione audio-visuale.

6.2 Sviluppi futuri

Lo step successivo è sicuramente quello dell'utilizzo dei dati biomedici per l'influenza degli effetti grafici, che resta un punto importante nel progetto. Similmente a quanto fatto per i dati audio, i valori che vengono attualmente letti tramite i sensori di ECG e GSR dovranno essere interpretati ed utilizzati dal sistema per riconoscere coerentemente lo stato emotivo del performer.

Per quanto riguarda lo specifico ambito dell'elaborazione dell'audio, gli sviluppi per il futuro sono molteplici: in primis l'aggiunta di nuove features audio sarà utile, non tanto per il controllo diretto dei visual, quanto per il miglioramento del rilevamento dello stato e delle transizioni, avvicinandoci sempre più ad un modello di mood completo.

I parametri in uscita da Dreamo sarebbero così già etichettati dal punto di vista emotivo, e sarebbe più semplice per chi lo utilizza creare grafiche interessanti, originali, e coerenti con la propria idea di performance.

Un importante feature è quella del modo musicale: seppur il calcolo non sia semplice ed i risultati degli algoritmi proposti in letteratura non siano sempre precisi, il modo è di fondamentale impatto sulla percezione emotiva della musica. Questo algoritmo è in fase di sviluppo (in particolare si tratta di un algoritmo basato sui chromagrammi) ed è prevista la sua implementazione nel prossimo futuro.

Anche dal punto di vista grafico ci sono molte idee: abbandonare il sistema scene/particelle e passare ad un meccanismo generativo più aperto ed organico potrebbe aumentare significativamente la particolarità e l'originalità di ogni performance, non essendo dipendente da grafiche pre-impostate.

Tuttavia questa strada potrebbe rivelarsi complessa e potrebbe togliere importanza all'artista grafico che si occupa della creazione degli effetti (più intelligente è la macchina, meno l'uomo è necessario).

È prevista inoltre la creazione inoltre di un database di lavori e parti di codice, al fine di aumentare il coinvolgimento in termini di community e migliorare lo scambio di informazioni, in linea con la filosofia open source che contraddistingue il mondo dei makers.

Bibliografia

1. Fab Lab. *Available at: https://en.wikipedia.org/wiki/Fab_lab*.
2. Shiffman, D., Fry, S. & Marsh, Z. *The nature of code*. (D. Shiffman, 2012).
3. Casey, M. A., Veltkamp, R., Goto, M. & Leman, M. Content-based music information retrieval: Current directions and future challenges. *Proceedings of the IEEE* 96.4 (2008): 668-696.
4. Tzanetakis, G. & Cook, P. Musical genre classification of audio signals. *IEEE Transactions on speech and* (2002).
5. Pampalk, E. *Islands of music: Analysis, organization, and visualization of music archives*. (na, 2001).
6. Fu, Z., Lu, G., Ting, K. M. & Zhang, D. A Survey of Audio-Based Music Classification and Annotation. *IEEE Trans. Multimedia* **13,** 303–319 (2011).
7. Downie, J. S. The music information retrieval evaluation exchange (2005–2007): A window into music information retrieval research. *Acoust. Sci. Technol.* **29,** 247–255 (2008).
8. Lerch, A. *An Introduction to Audio Content Analysis: Applications in Signal Processing and Music Informatics*. (John Wiley & Sons, 2012).
9. Giannakopoulos, T. & Pikrakis, A. *Introduction to Audio Analysis: A MATLAB® Approach*. (Academic Press, 2014).
10. Tzanetakis, G. & Cook, P. Marsyas: A framework for audio analysis. *Organised sound* **4,** 169–175 (2000).
11. Brossier, P. M. The aubio library at mirex 2006. *Synthesis* (2006).
12. Lartillot, O. & Toiviainen, P. A Matlab toolbox for musical feature extraction from audio. in *International Conference on Digital Audio Effects* 237–244 (2007).
13. Six, J. Digital Sound Processing and Java. (2012).
14. Downie, X., Laurier, C. & Ehmann, M. The 2007 MIREX audio mood classification task: Lessons learned. in *Proc. 9th Int. Conf. Music Inf. Retrieval* 462–467 (2008).
15. Bukowski, M. Music Mood Classification: From a Music Piece to a Computed Mood.

(2015).

16. Juslin, P. N. & Laukka, P. Expression, Perception, and Induction of Musical Emotions: A Review and a Questionnaire Study of Everyday Listening. *Journal of New Music Research* **33,** 217–238 (2004).
17. Kim, Y. E. *et al.* Music emotion recognition: A state of the art review. in *Proc. ISMIR* 255–266 (2010).
18. Laurier, C., Meyers, O., Serrà, J. & Blech, M. Indexing music by mood: design and integration of an automatic content-based annotator. *Multimed. Tools Appl.* (2010).
19. Lu, L., Liu, D. & Zhang, H.-J. Automatic mood detection and tracking of music audio signals. *IEEE Trans. Audio Speech Lang. Processing* **14,** 5–18 (2006).
20. Li, T. & Ogihara, M. Detecting emotion in music. (2003).
21. Bigand, E., Vieillard, S., Madurell, F., Marozeau, J. & Dacquet, A. Multidimensional scaling of emotional responses to music: The effect of musical expertise and of the duration of the excerpts. *Cognition and Emotion* **19,** 1113–1139 (2005).
22. Mion, L. & Poli, G. D. Score-Independent Audio Features for Description of Music Expression. *IEEE Trans. Audio Speech Lang. Processing* **16,** 458–466 (2008).
23. Yang, Y. H., Lin, Y. C., Su, Y. F. & Chen, H. H. A Regression Approach to Music Emotion Recognition. *IEEE Trans. Audio Speech Lang. Processing* **16,** 448–457 (2008).
24. Bohnacker, H., Gross, B., Laub, J. & Lazzeroni, C. *Generative Design: Visualize, Program, and Create with Processing*. (Princeton Architectural Press, 2012).
25. Soraghan, S., Renaud, A. & Supper, B. A perceptually motivated visualisation paradigm for musical timbre. in *Proceedings of the conference on Electronic Visualisation and the Arts* 53–60 (British Computer Society, 2016).
26. Hiraga, R. & Matsuda, N. Graphical expression of the mood of music. in *2004 IEEE International Conference on Multimedia and Expo (ICME) (IEEE Cat. No.04TH8763)* **3,** 2035–2038 Vol.3 (2004).
27. Foote, J. Visualizing Music and Audio Using Self-similarity. in *Proceedings of the Seventh ACM International Conference on Multimedia (Part 1)* 77–80 (ACM, 1999).
28. Bragg, D. Quantification and Display of Emotions in Music. (2010).

29. Vieira, M. F. G. Interactive music visualization: implementation, realization and evaluation. (2014).
30. Moritz, W. *Optical Poetry: The Life and Work of Oskar Fischinger*. (Indiana University Press, 2004).
31. Fischinger, O. *Kreise*. (1933).
32. Fischinger, O. *Composition in Blue*. (1935).
33. 'Walter Ruttmann'. *Wikipedia* Available at: https://en.wikipedia.org/wiki/Walter_Ruttmann.
34. Ruttmann, W. *Lichtspiel: Opus I*. (1921).
35. Ruttmann, W. *Lichtspiel: Opus II*. (1921).
36. Searle, A. Oskar Fischinger: the animation wizard who angered Walt Disney and the Nazis. *The Guardian* (2013).
37. Kurokawa, R. *Rheo: 5 horizons (audiovisual installation)*. (2010).
38. Li, T., Ogihara, M. & Li, Q. A comparative study on content-based music genre classification. *Proceedings of the 26th annual international ACM* (2003).
39. Sharma, S. & Jadon, R. S. Mood Based Music Classification. *IJISET* **1,** (2014).
40. Peeters, G. *{A large set of audio features for sound description (similarity and classification) in the CUIDADO project}*. (2004).
41. Gouyon, F., Pachet, F., Delerue, O. & Others. On the use of zero-crossing rate for an application of classification of percussive sounds. in *Proceedings of the COST G-6 conference on Digital Audio Effects (DAFX-00), Verona, Italy* (2000).
42. Bhat, A. S., Amith, V. S. & Prasad, N. S. An efficient classification algorithm for music mood detection in Western and Hindi music using audio feature extraction. *Signal and Image* (2014).
43. Tzanetakis, G., Essl, G. & Cook, P. Audio analysis using the discrete wavelet transform. in *Proc. Conf. in Acoustics and Music Theory Applications* (2001).
44. Barry, D., Fitzgerald, D., Coyle, E. & Lawlor, B. Drum source separation using percussive feature detection and spectral modulation. 13–17 (2005).
45. Patin, F. Beat detection algorithms. *Compare A Journal Of Comparative Education*

(2003).

46. John Varney, Biljana Labovic, Peyton Skyler, Alex Gendler. *A different way to visualize rhythm*. (TED-Ed, 2014).
47. Hinn, D. M. The effect of the major and minor mode in music as a mood induction procedure. (Virginia Tech, 1996).
48. Tolonen, T. & Karjalainen, M. A computationally efficient multipitch analysis model. *IEEE Trans. Audio Speech Lang. Processing 8, 708–71*6 (2000).
49. D'Andrade, R. & Egan, M. The colors of emotion. *Am. Ethnol.* **1,** 49–63 (1974).
50. NAz, K. & Epps, H. Relationship between color and emotion: A study of college students. *Coll. Stud. J.* (2004).
51. Ou, L.-C., Luo, M. R., Woodcock, A. & Wright, A. A study of colour emotion and colour preference. Part I: Colour emotions for single colours. *Color Res. Appl.* **29,** 232–240 (2004).
52. Soraghan, S., Renaud, A. & Supper, B. A perceptually motivated visualisation paradigm for musical timbre. in *Proceedings of the conference on Electronic Visualisation and the Arts* 53–60 (British Computer Society, 2016).
53. Studio, C. V. *Spiral Galaxy: a computer graphics music video*. (1975-1985).
54. Abel, T. M. *Circular Movement (music video)*.
55. Perlin, K. Improving Noise. in *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques* 681–682 (ACM, 2002).

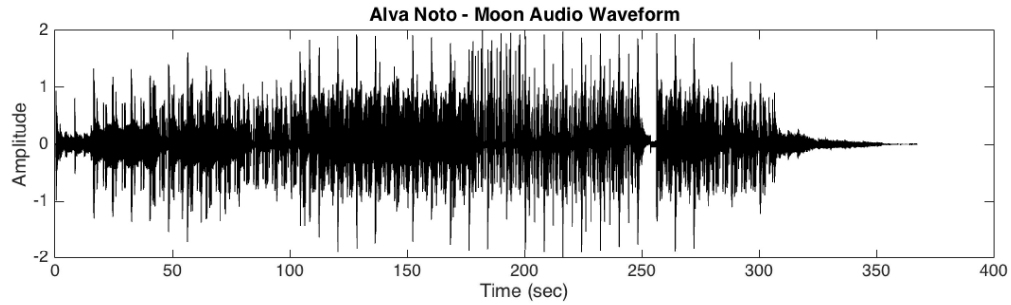
Appendice A

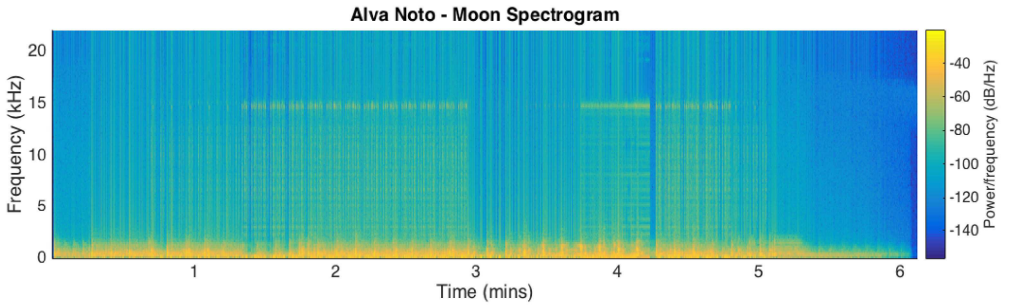
Andamento temporale delle features acustiche

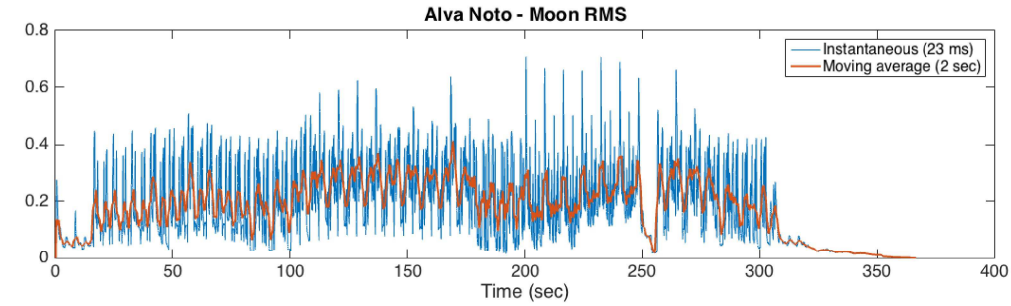
In quest'appendice sono riportati i grafici dell'andamento temporale delle 6 features estratte, della forma d'onda
e dello spettrografo di 11 brani significativi che facevano parte del test set.

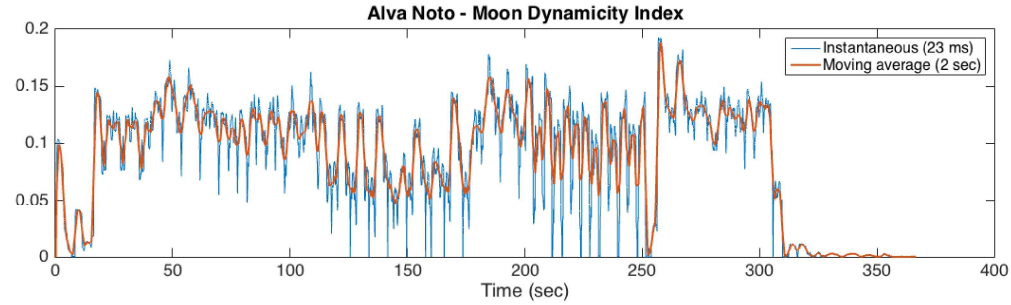
- Alva Noto & Ryuchi Sakamoto - Moon (elettronica/ambient, 2005)
- Atoms For Peace - Dropped (indietronica/electronic-pop, 2013)
- Fat Freddy's Drop - Shiverman (dub/reggae, 2009)
- Heliocentrics - Joyride (psychedelic/ethnic/jazz, 2007)
- Infected Mushroom - IM The Supervisor (psy-trance, 2004)
- Massive Attack - Angel (elettronica/trip-hop, 1998)
- Nathan Fake - The Sky Was Pink (idm/indietronica, 2006)
- Red Snapper - Get Some Sleep Tiger (acid jazz/breakbeat, 1996)
- Tinariwen - Ittus (desert blues/world, 2017)
- Tommy Guerrero - Thank You MK (lo-fi/chill out, 2003)
- Dreamo Track (traccia test)

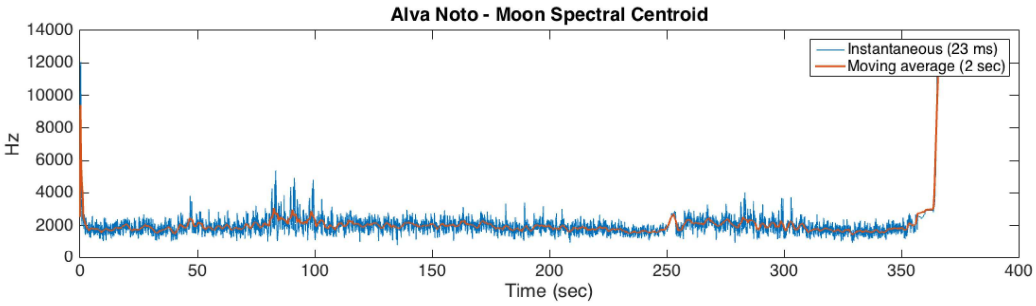
A.1 Alva Noto - Moon

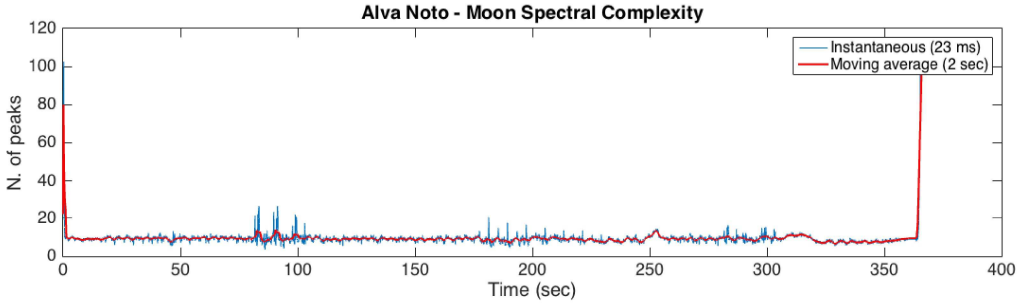


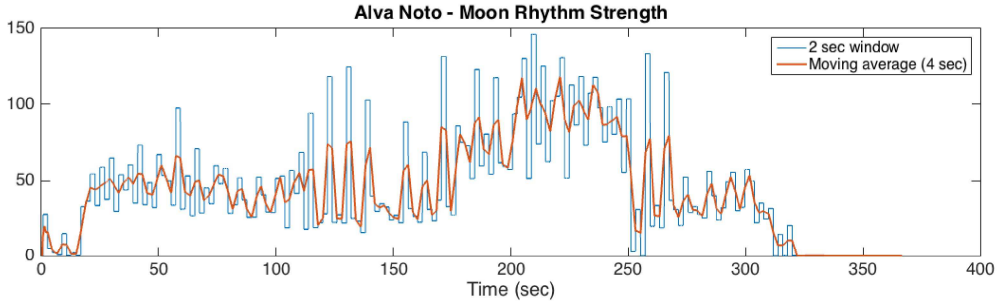


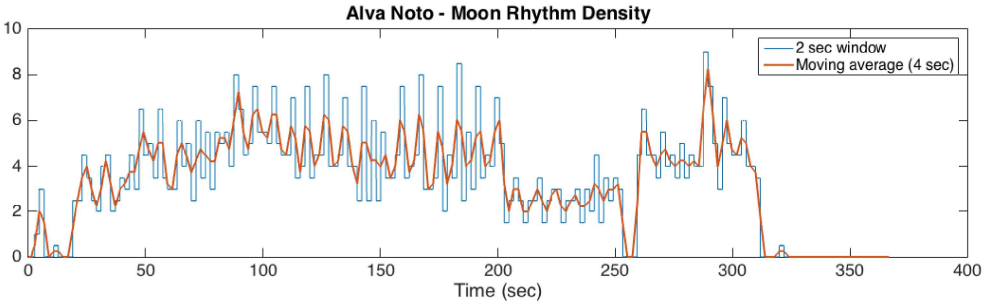




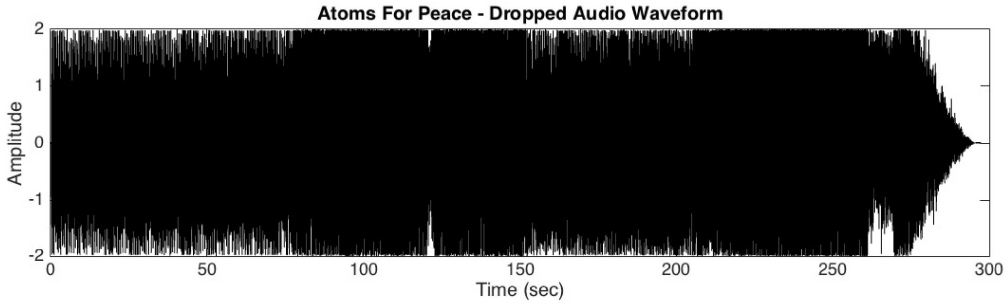

Alva Noto - Moon Spectral Centroid
Instantaneous (23 ms)
Moving average (2 sec)
Hz
Time (sec)

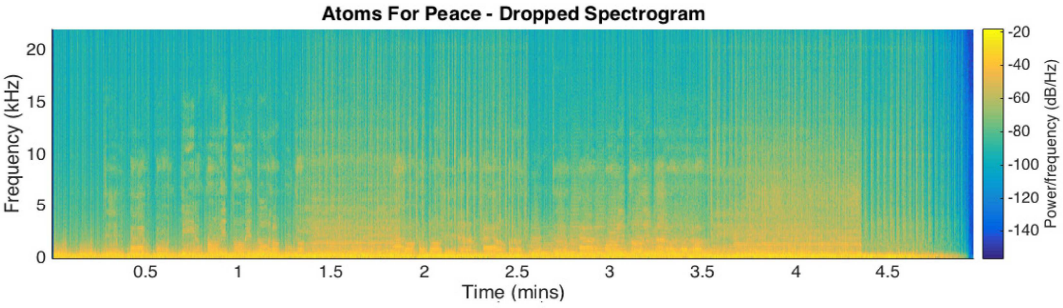

Alva Noto - Moon Spectral Complexity
Instantaneous (23 ms)
Moving average (2 sec)
N. of peaks
Time (sec)

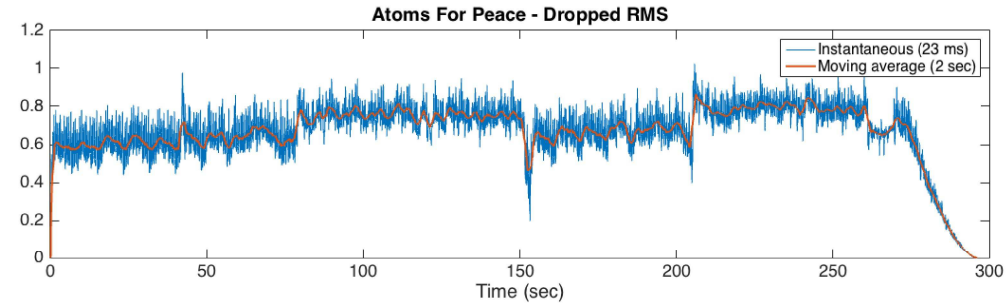

Alva Noto - Moon Rhythm Strength
2 sec window
Moving average (4 sec)
Time (sec)

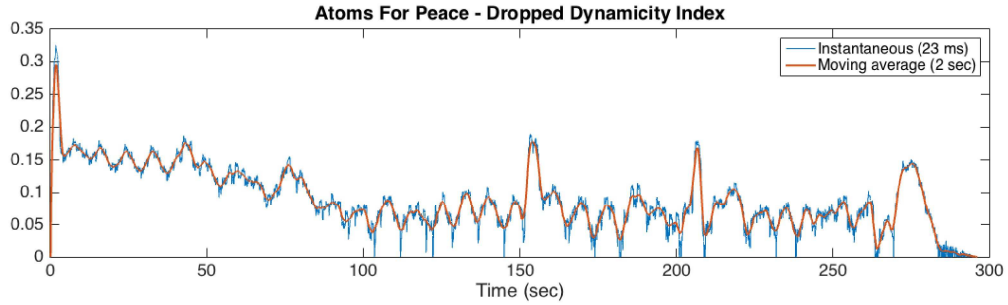

Alva Noto - Moon Rhythm Density
2 sec window
Moving average (4 sec)
Time (sec)

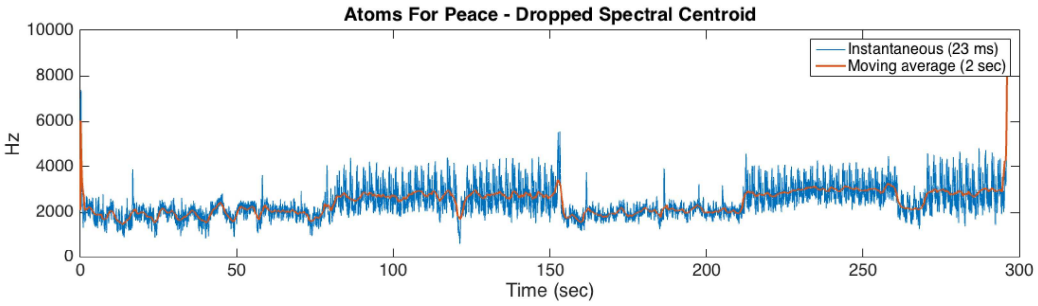
A.2 Atoms For Peace - Dropped

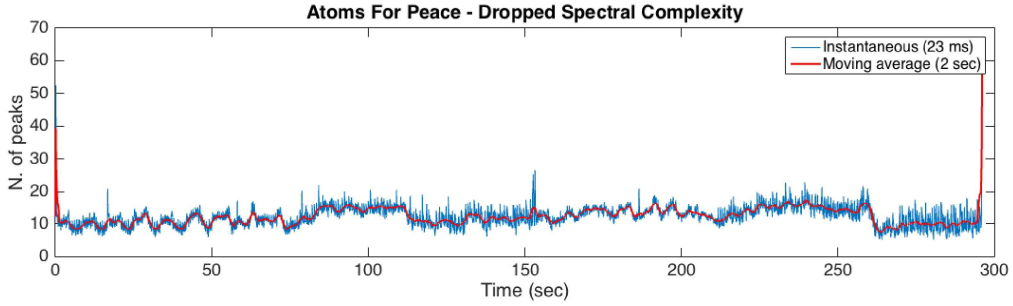


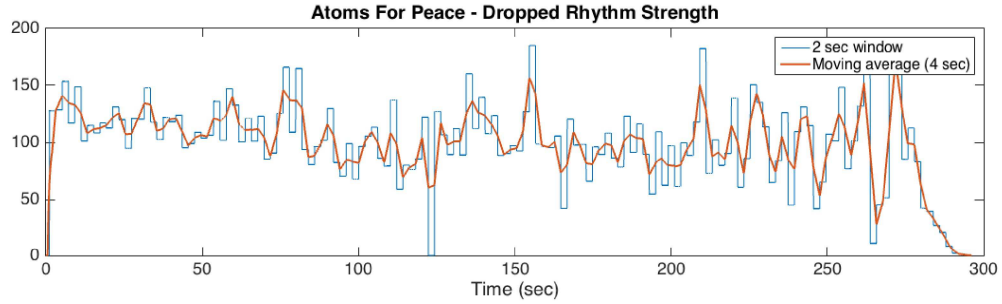


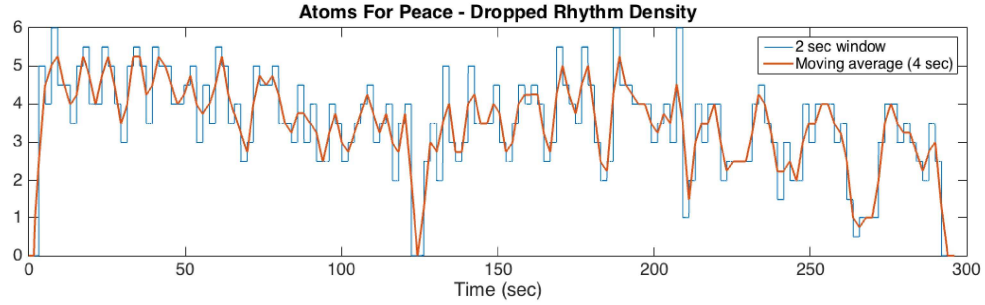




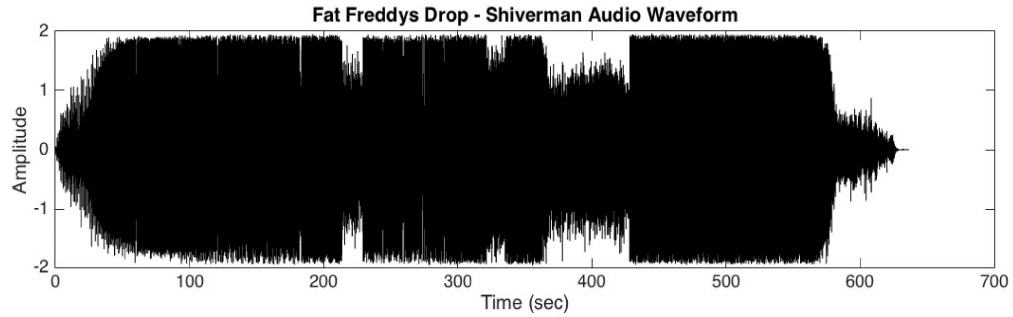

Atoms For Peace - Dropped Spectral Centroid
Instantaneous (23 ms)
Moving average (2 sec)
Hz
Time (sec)

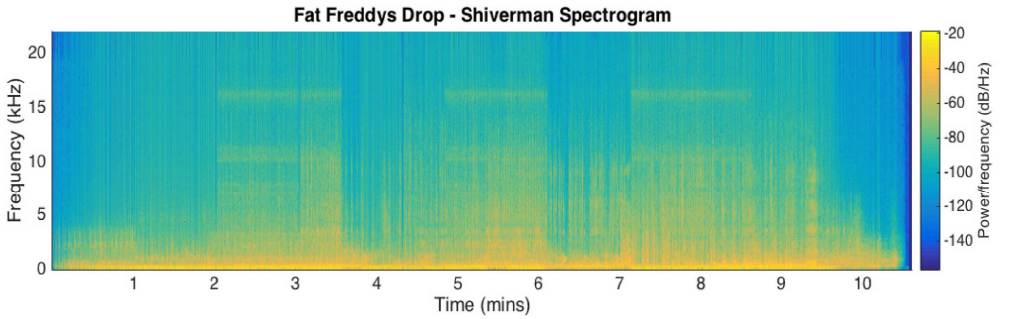

Atoms For Peace - Dropped Spectral Complexity
Instantaneous (23 ms)
Moving average (2 sec)
N. of peaks
Time (sec)

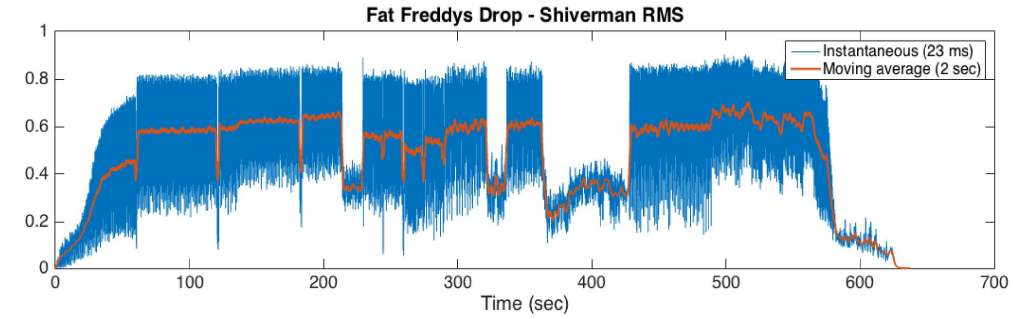

Atoms For Peace - Dropped Rhythm Strength
2 sec window
Moving average (4 sec)
Time (sec)

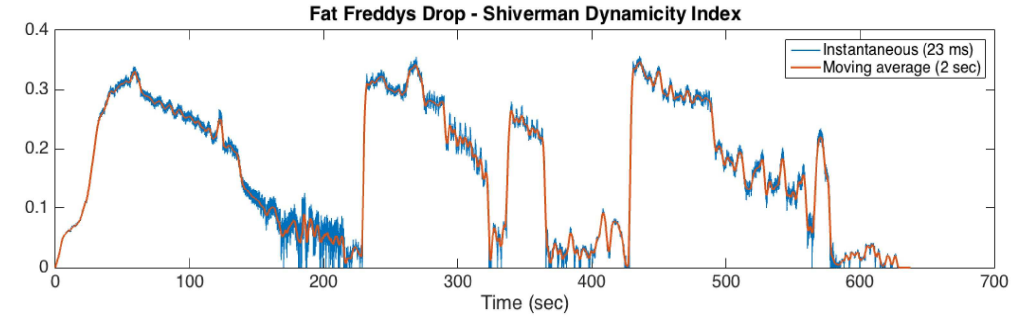

Atoms For Peace - Dropped Rhythm Density
2 sec window
Moving average (4 sec)
Time (sec)

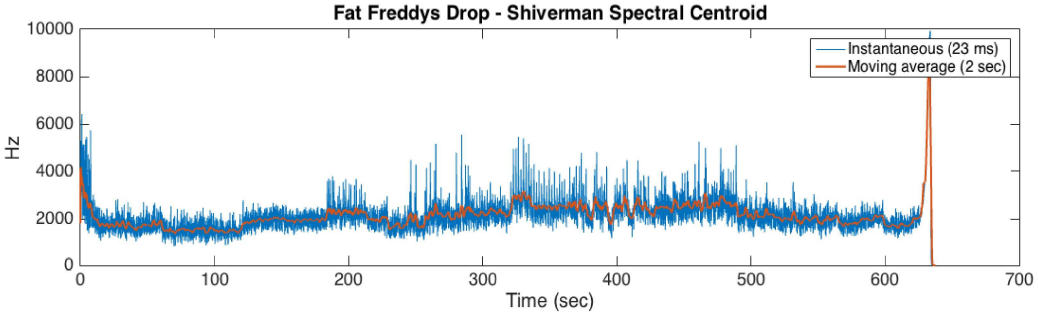
A.3 Fat Freddys Drop - Shiverman

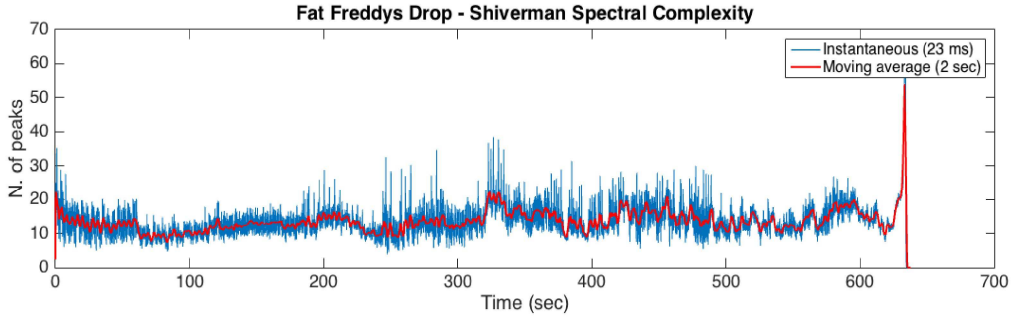


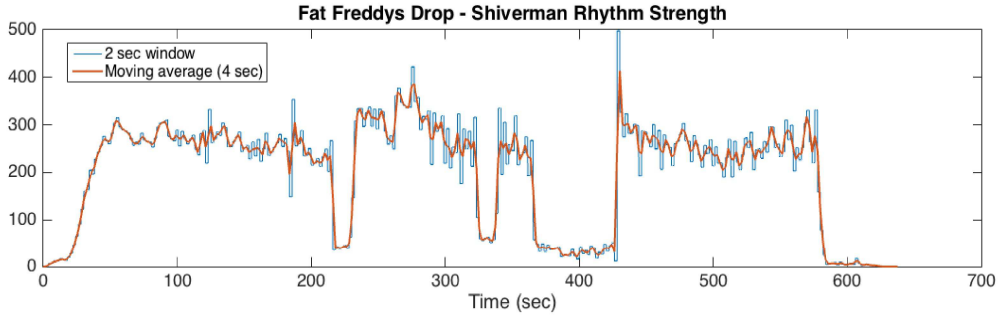


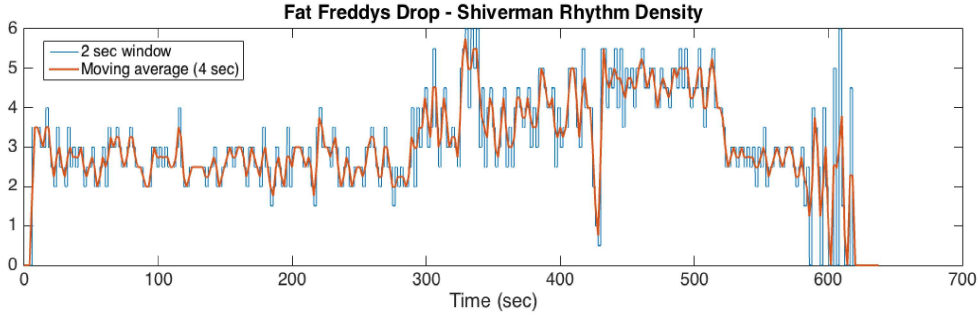




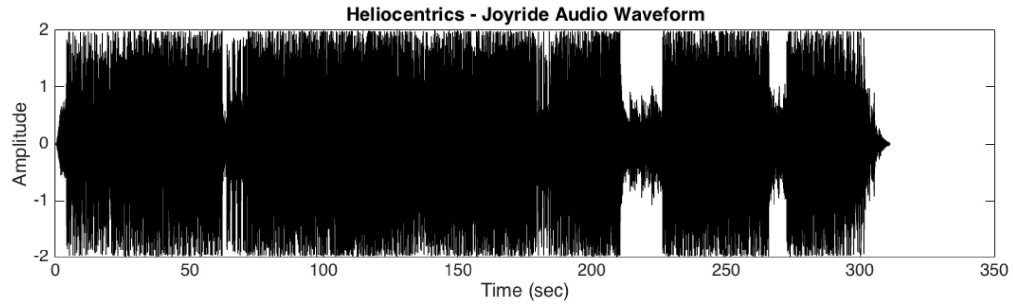

Fat Freddys Drop - Shiverman Spectral Centroid
Instantaneous (23 ms)
Moving average (2 sec)
Hz
Time (sec)

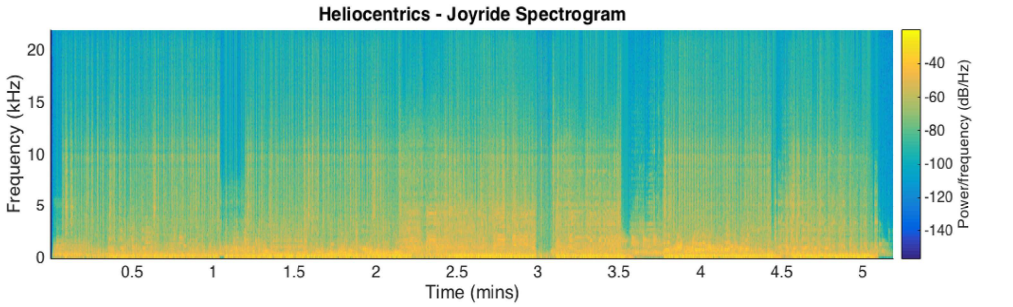

Fat Freddys Drop - Shiverman Spectral Complexity
Instantaneous (23 ms)
Moving average (2 sec)
N. of peaks
Time (sec)

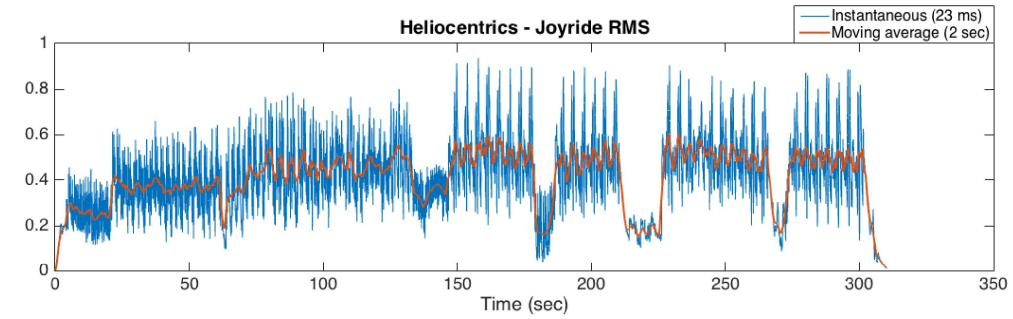

Fat Freddys Drop - Shiverman Rhythm Strength
2 sec window
Moving average (4 sec)
Time (sec)

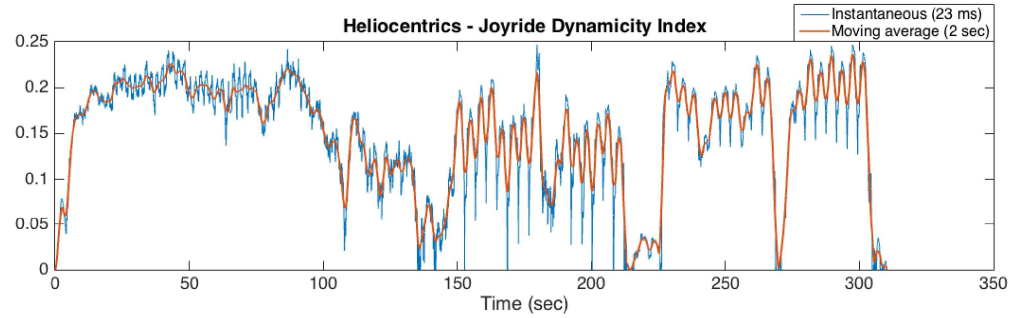

Fat Freddys Drop - Shiverman Rhythm Density
2 sec window
Moving average (4 sec)
Time (sec)

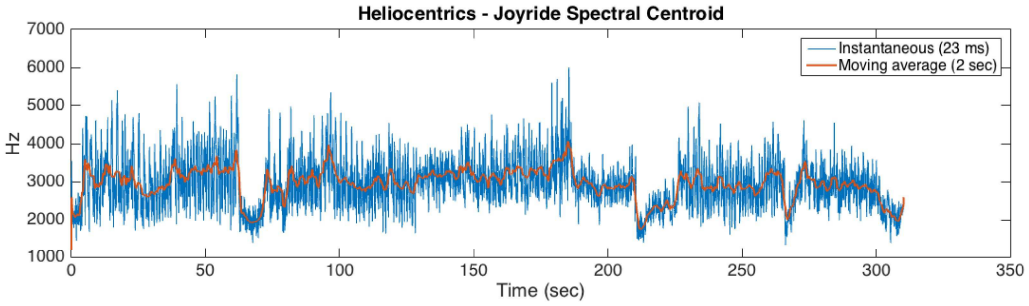
A.4 Heliocentrics - Joyride

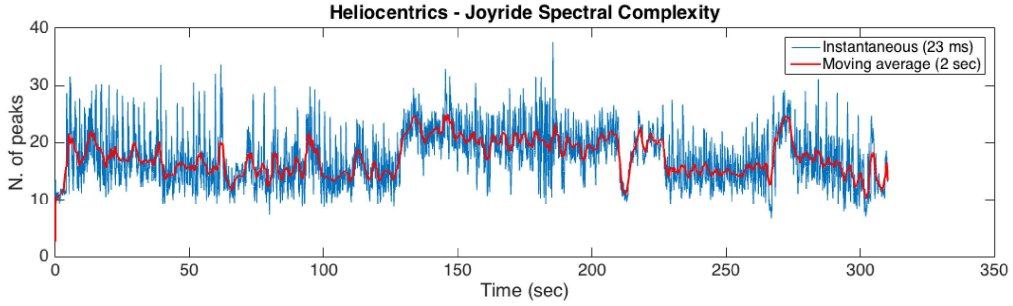


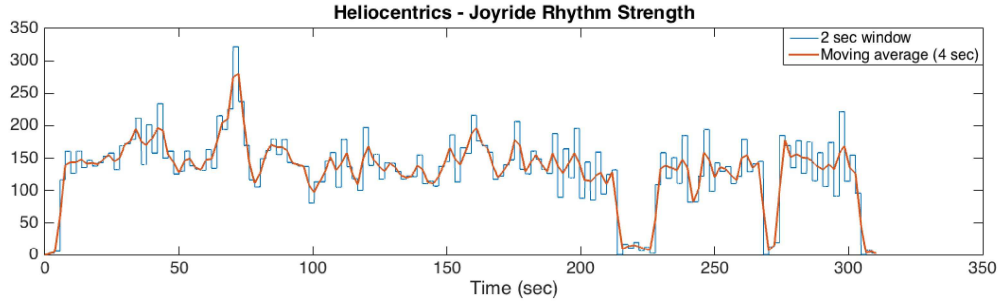


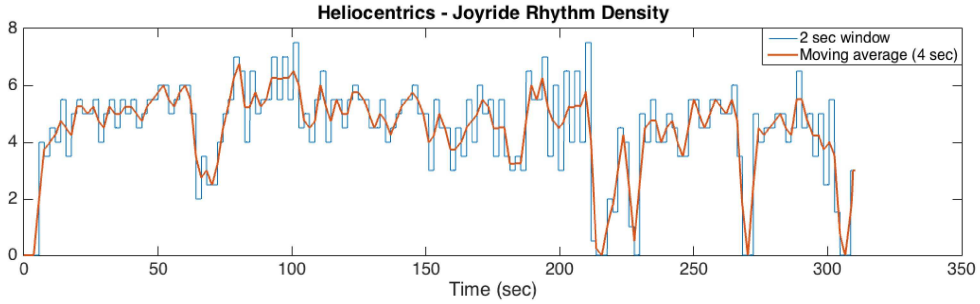




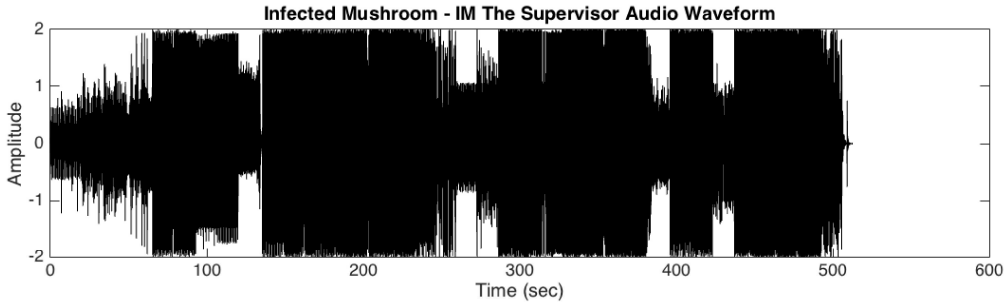

Heliocentrics - Joyride Spectral Centroid
Instantaneous (23 ms)
Moving average (2 sec)
Hz
Time (sec)

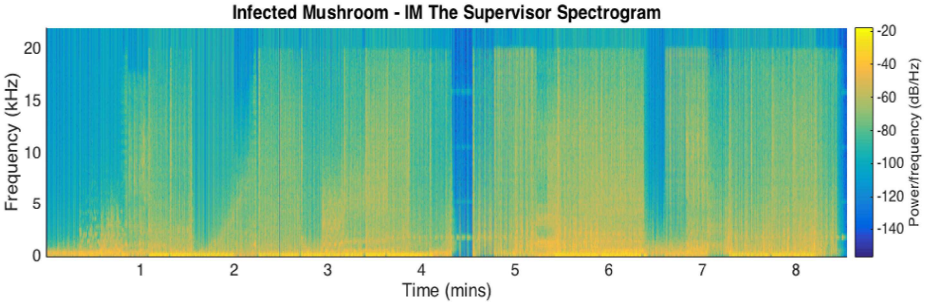

Heliocentrics - Joyride Spectral Complexity
Instantaneous (23 ms)
Moving average (2 sec)
N. of peaks
Time (sec)

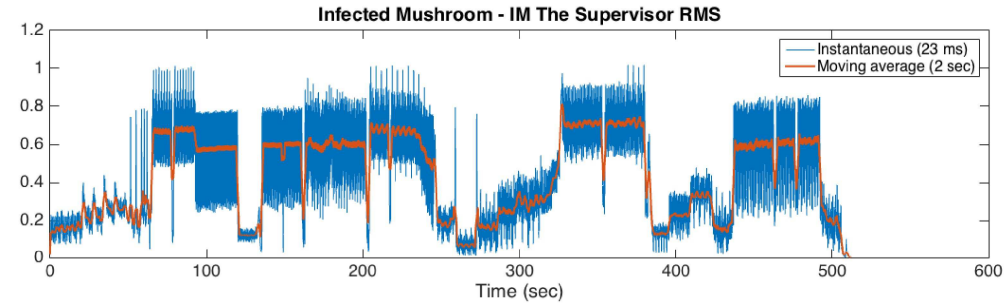

Heliocentrics - Joyride Rhythm Strength
2 sec window
Moving average (4 sec)
Time (sec)

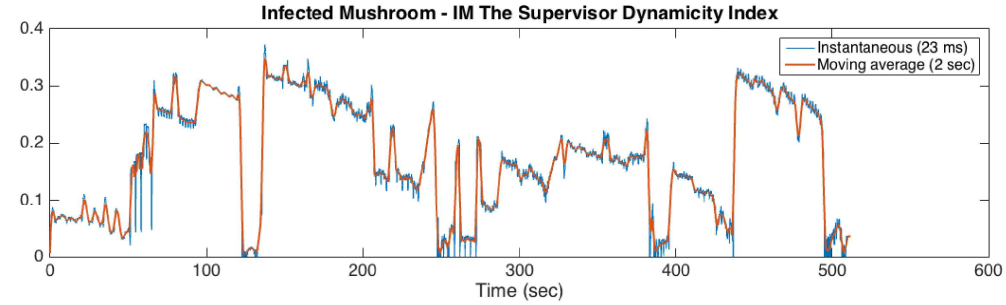

Heliocentrics - Joyride Rhythm Density
2 sec window
Moving average (4 sec)
Time (sec)

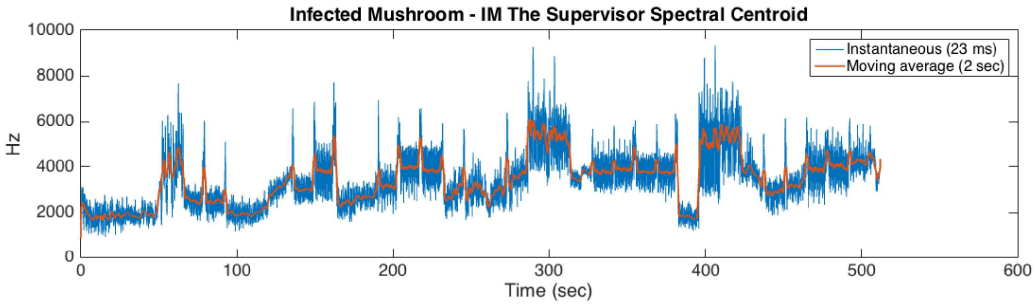
A.5 Infected Mushroom - IM The Supervisor

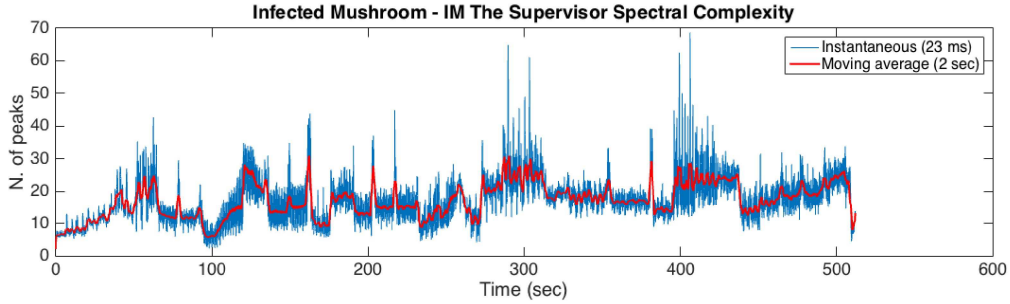


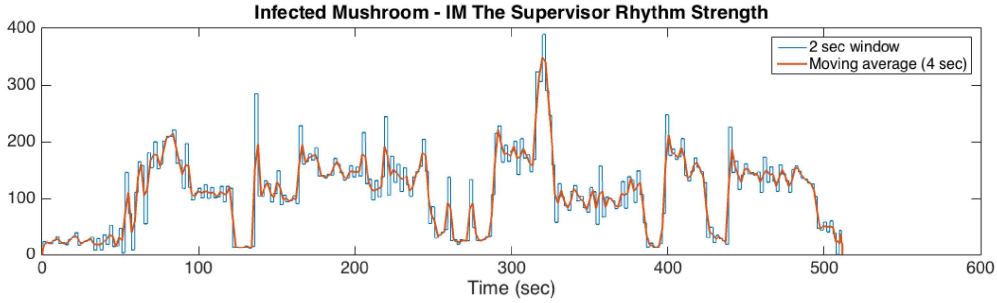


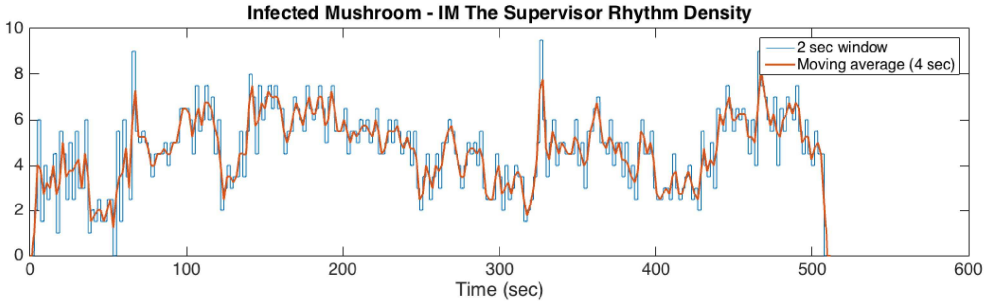




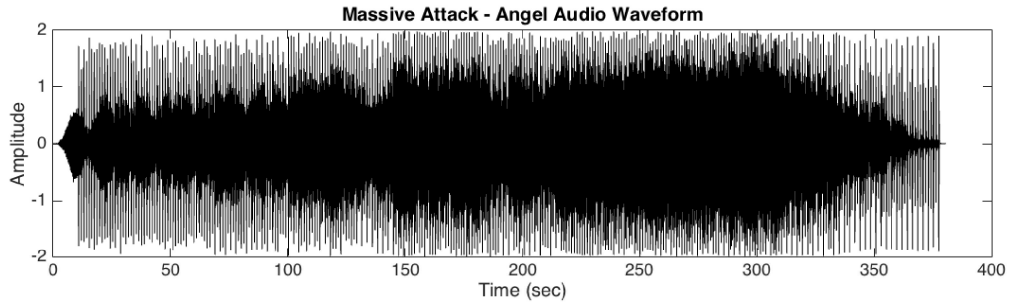

Infected Mushroom - IM The Supervisor Spectral Centroid
Instantaneous (23 ms)
Moving average (2 sec)
Hz
Time (sec)

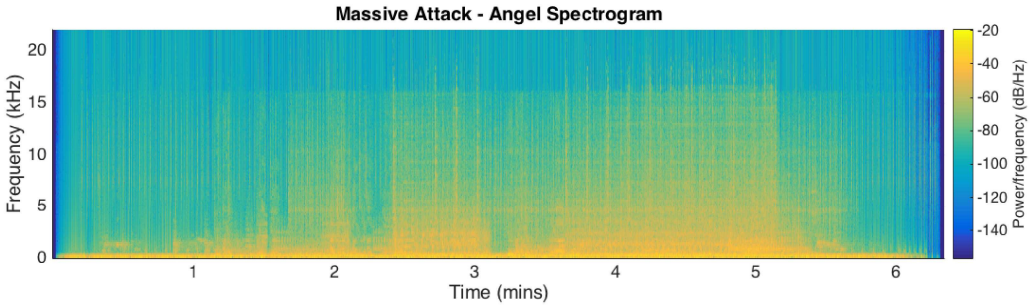

Infected Mushroom - IM The Supervisor Spectral Complexity
Instantaneous (23 ms)
Moving average (2 sec)
N. of peaks
Time (sec)

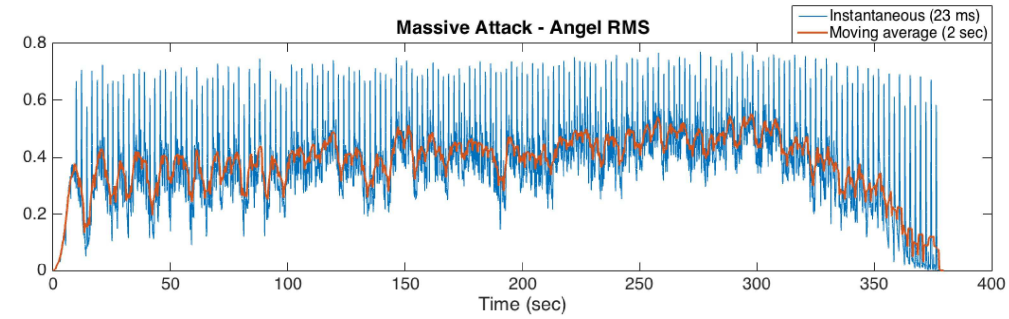

Infected Mushroom - IM The Supervisor Rhythm Strength
2 sec window
Moving average (4 sec)
Time (sec)

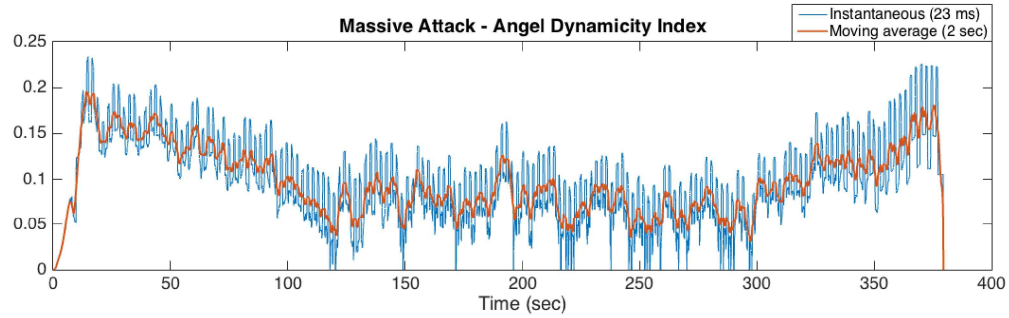

Infected Mushroom - IM The Supervisor Rhythm Density
2 sec window
Moving average (4 sec)
Time (sec)

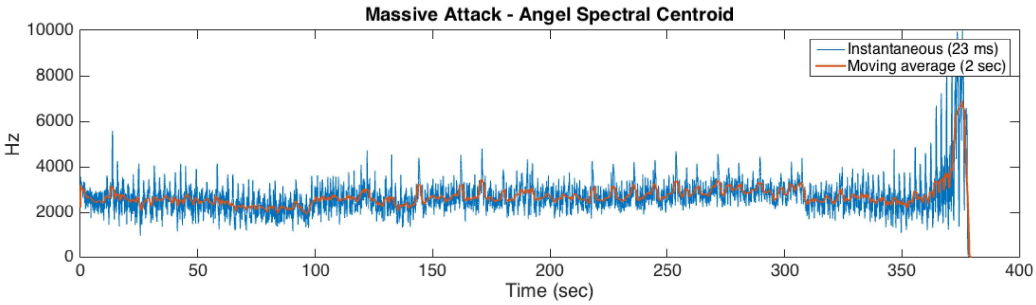
A.6 Massive Attack - Angel

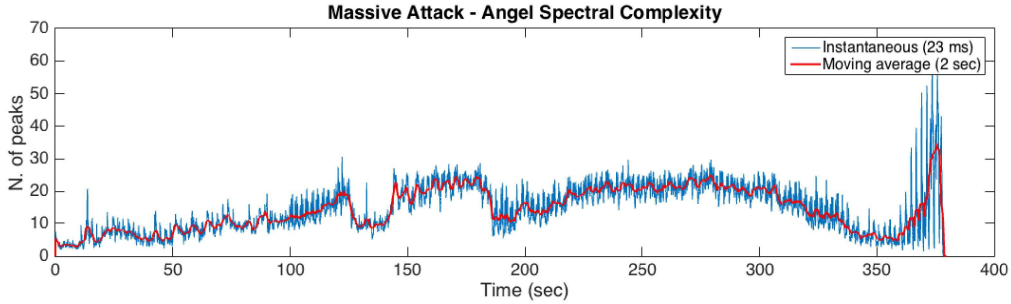


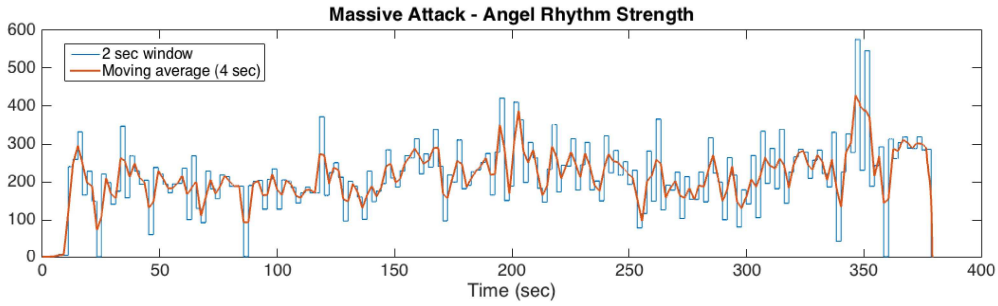


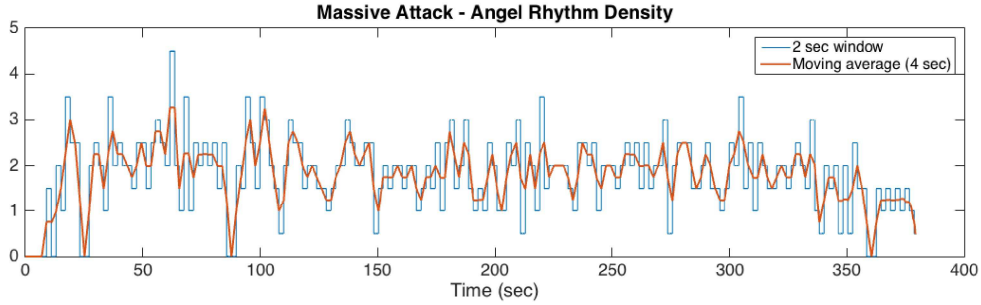




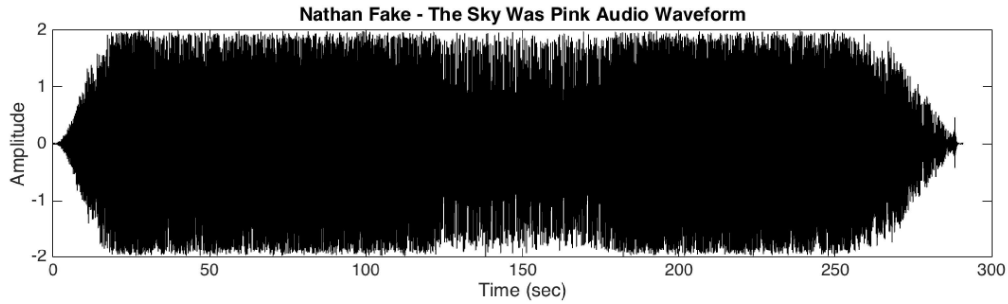

Massive Attack - Angel Spectral Centroid
Instantaneous (23 ms)
Moving average (2 sec)
Hz
Time (sec)

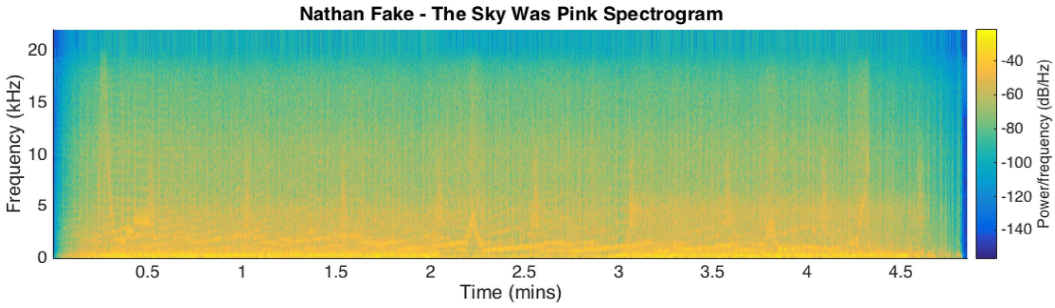

Massive Attack - Angel Spectral Complexity
Instantaneous (23 ms)
Moving average (2 sec)
N. of peaks
Time (sec)

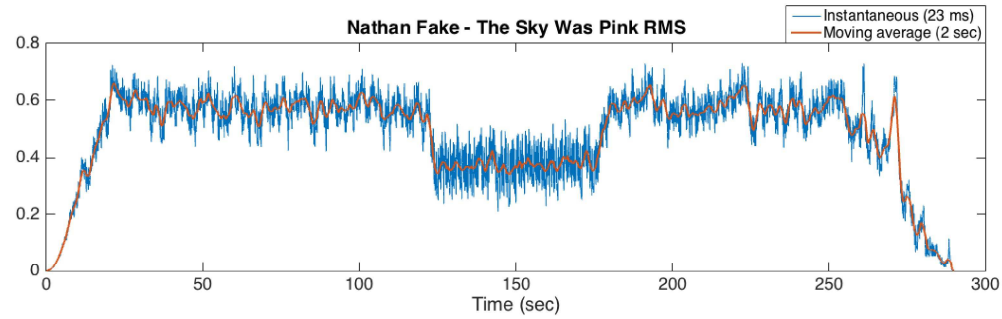

Massive Attack - Angel Rhythm Strength
2 sec window
Moving average (4 sec)
Time (sec)

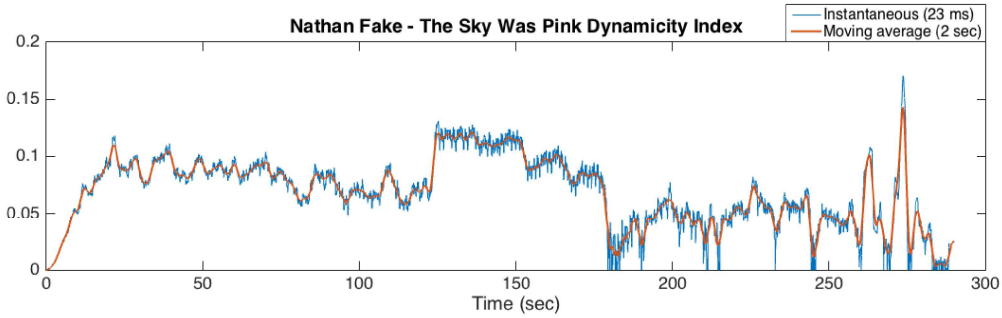

Massive Attack - Angel Rhythm Density
2 sec window
Moving average (4 sec)
Time (sec)

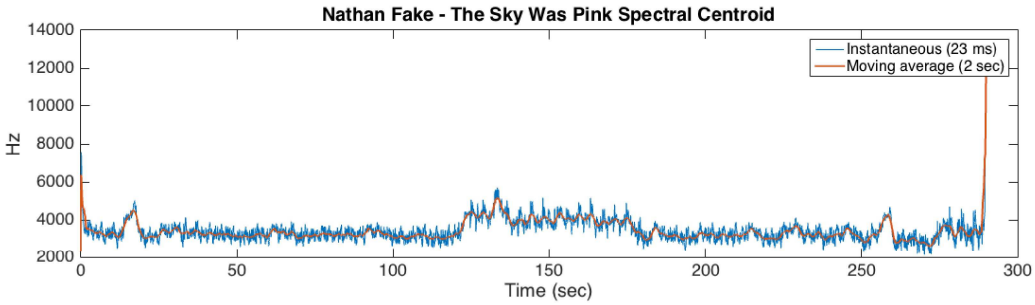
A.7 Nathan Fake - The Sky Was Pink

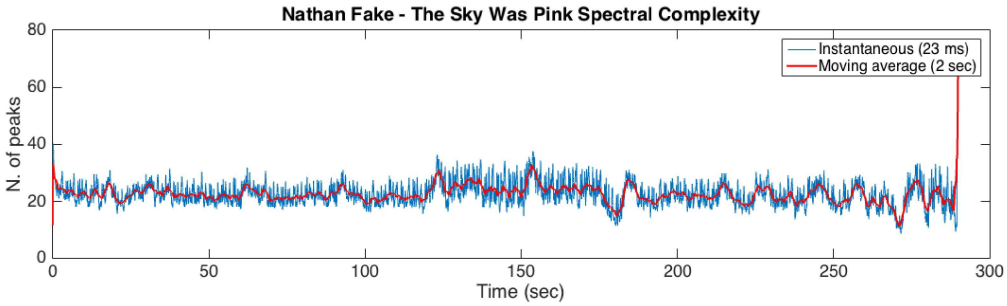


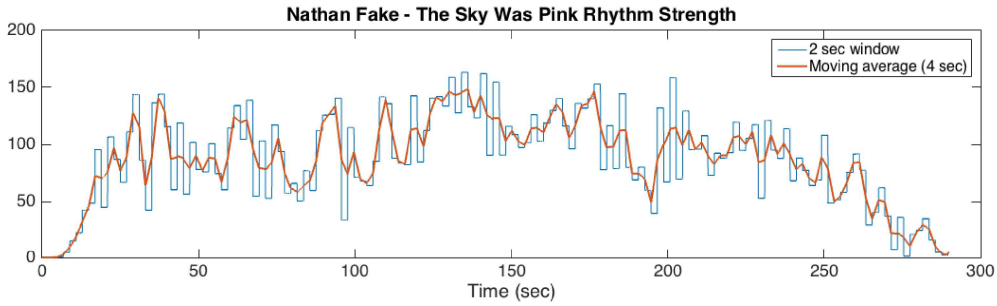


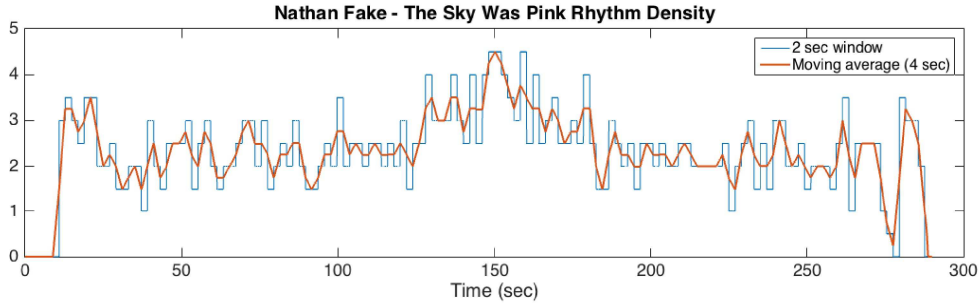




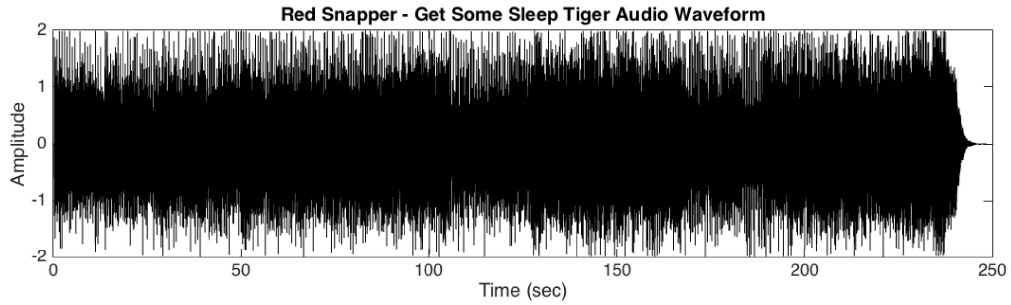

Nathan Fake - The Sky Was Pink Spectral Centroid
Instantaneous (23 ms)
Moving average (2 sec)
Hz
Time (sec)

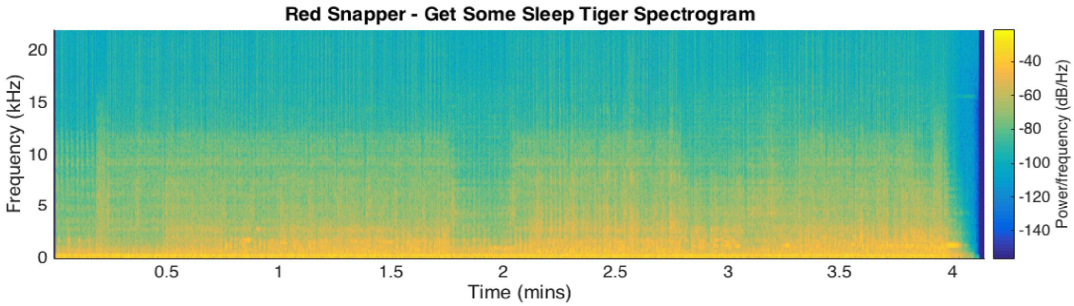

Nathan Fake - The Sky Was Pink Spectral Complexity
Instantaneous (23 ms)
Moving average (2 sec)
N. of peaks
Time (sec)

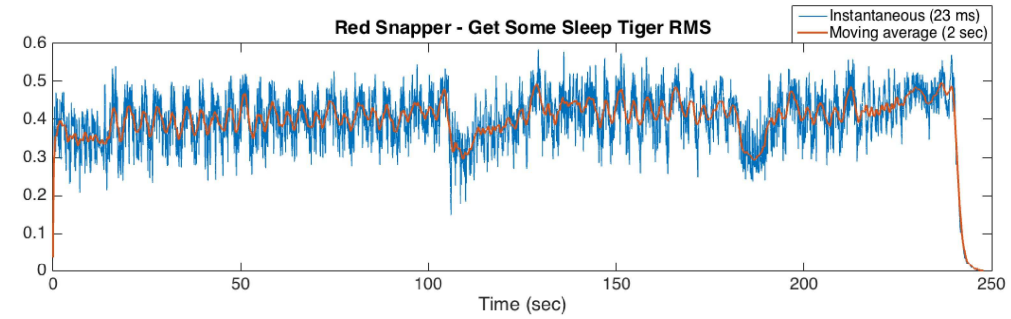

Nathan Fake - The Sky Was Pink Rhythm Strength
2 sec window
Moving average (4 sec)
Time (sec)

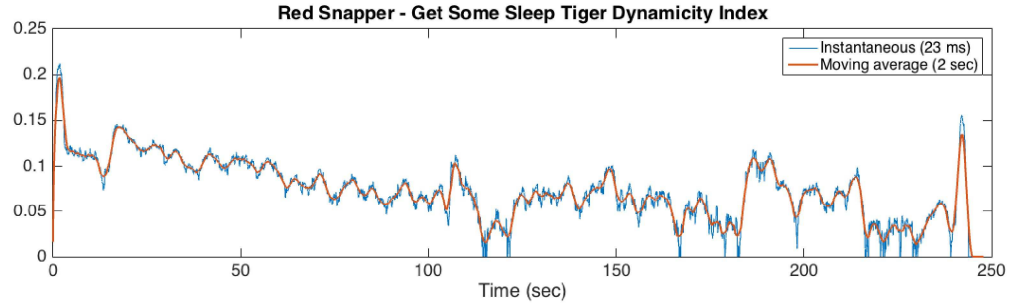

Nathan Fake - The Sky Was Pink Rhythm Density
2 sec window
Moving average (4 sec)
Time (sec)

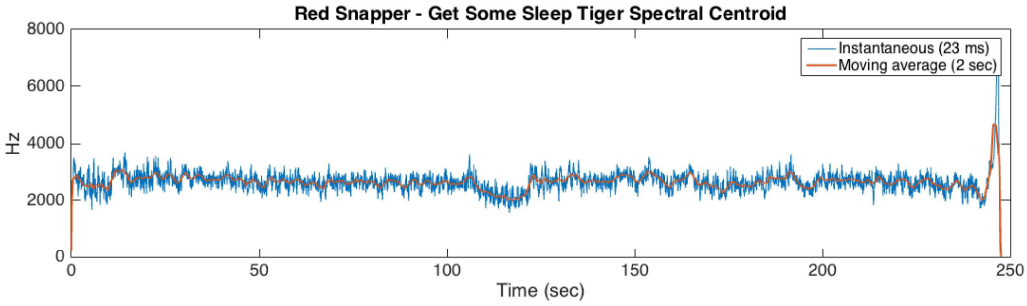
A.8 Red Snapper - Get Some Sleep Tiger

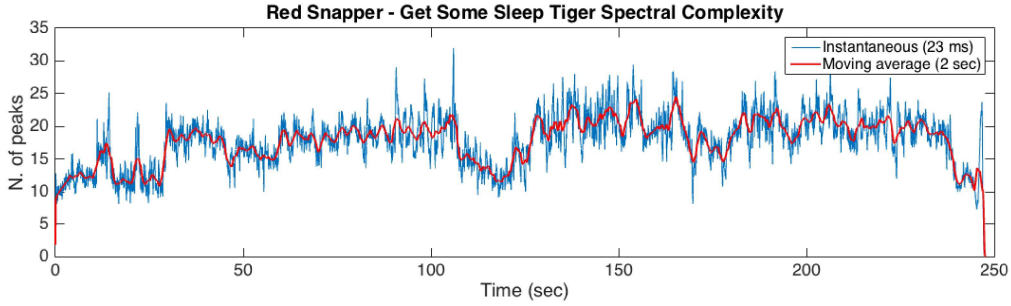


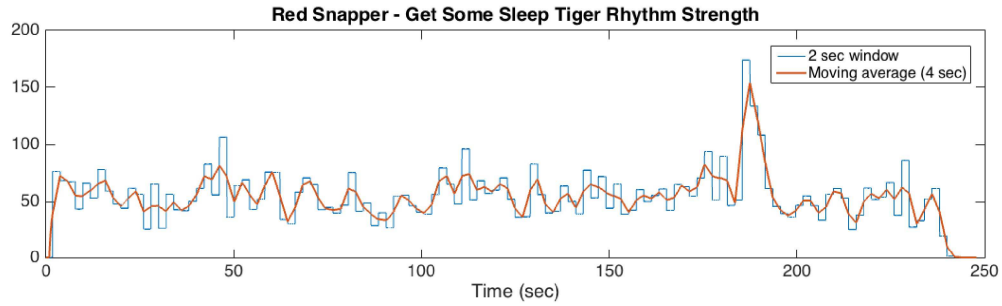


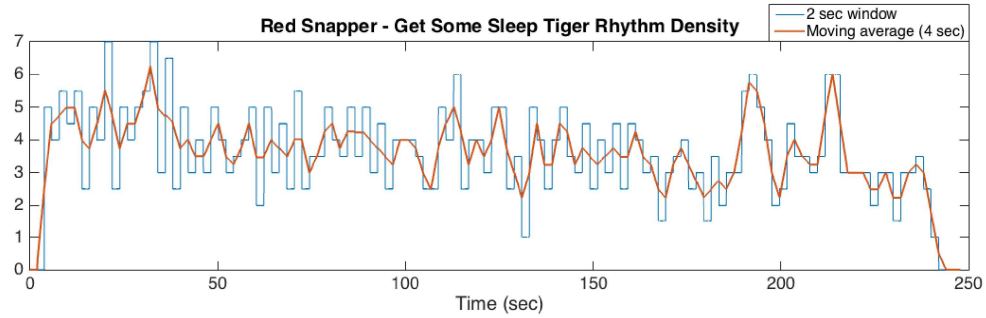




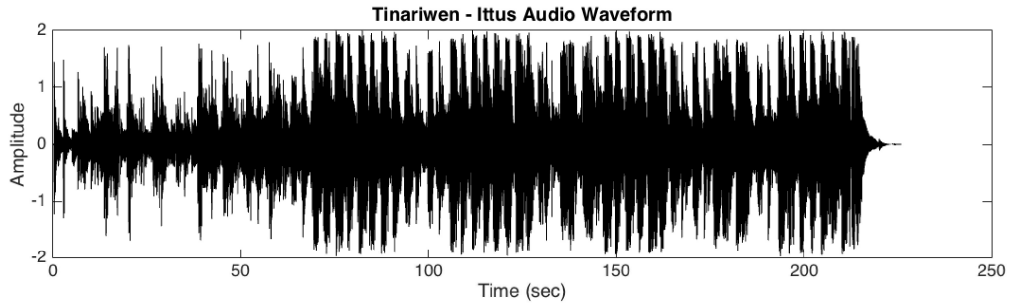

Red Snapper - Get Some Sleep Tiger Spectral Centroid
Instantaneous (23 ms)
Moving average (2 sec)
Hz
8000
6000
4000
2000
0
0
50
100
150
200
250
Time (sec)

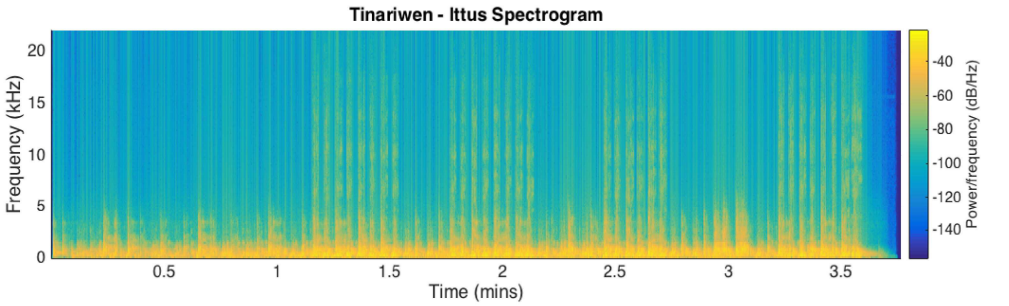

Red Snapper - Get Some Sleep Tiger Spectral Complexity
Instantaneous (23 ms)
Moving average (2 sec)
N. of peaks
35
30
25
20
15
10
5
0
0
50
100
150
200
250
Time (sec)

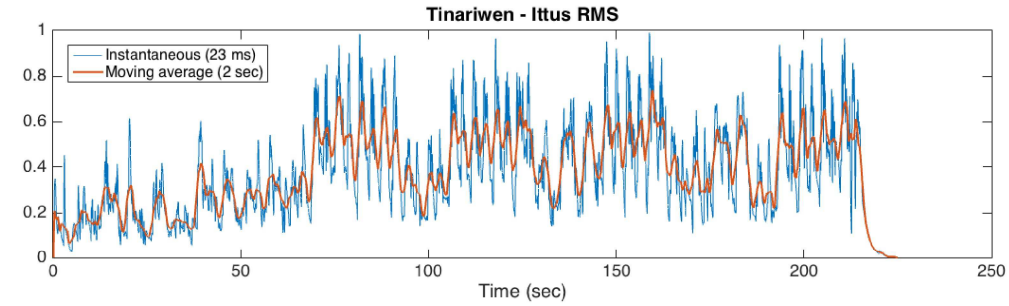

Red Snapper - Get Some Sleep Tiger Rhythm Strength
2 sec window
Moving average (4 sec)
200
150
100
50
0
0
50
100
150
200
250
Time (sec)

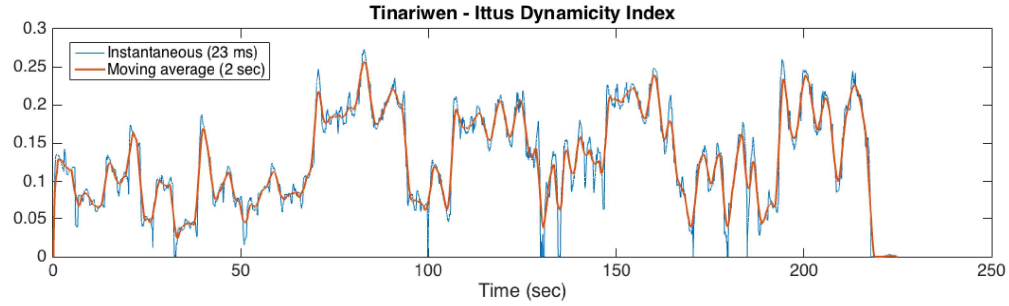

Red Snapper - Get Some Sleep Tiger Rhythm Density
2 sec window
Moving average (4 sec)
7
6
5
4
3
2
1
0
0
50
100
150
200
250
Time (sec)

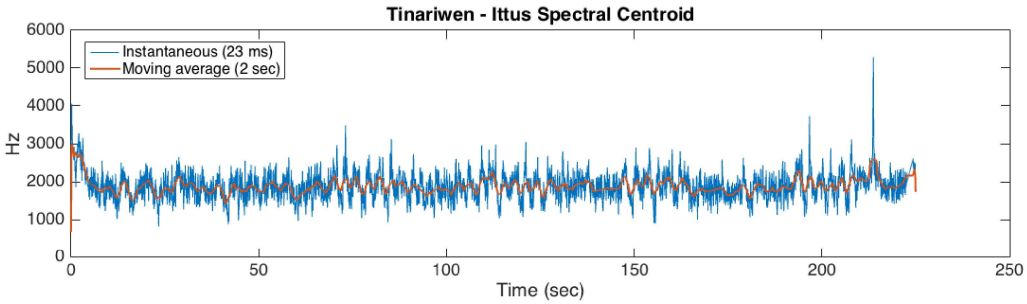
A.9 Tinariwen - Ittus

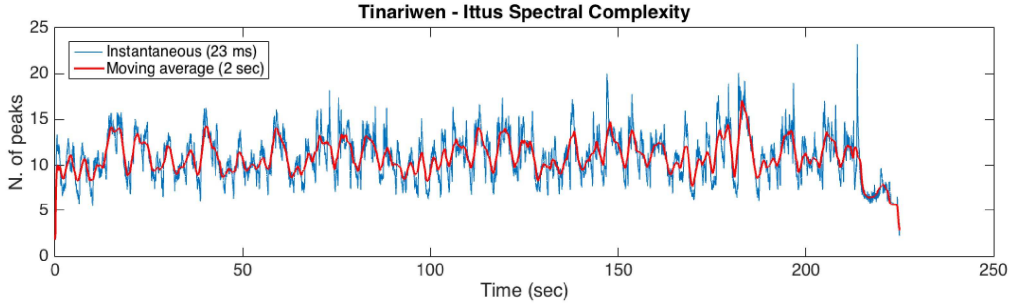


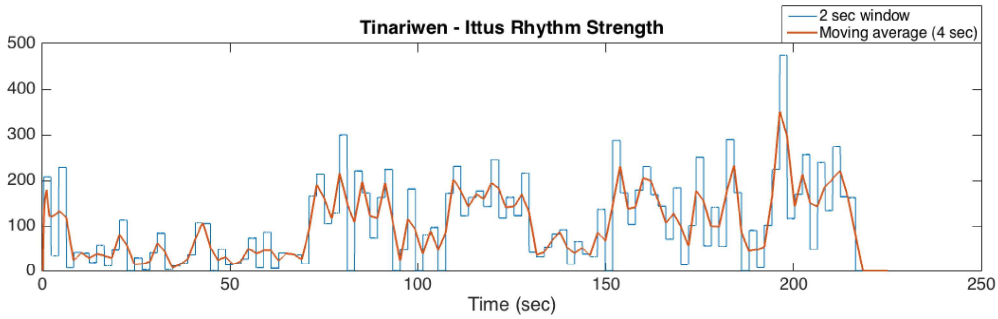


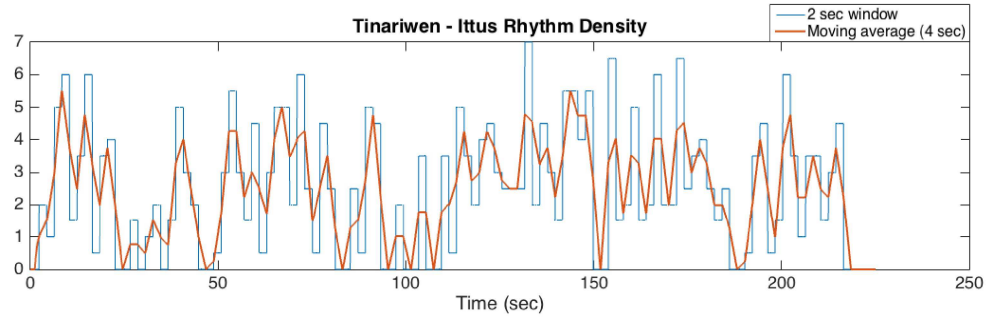




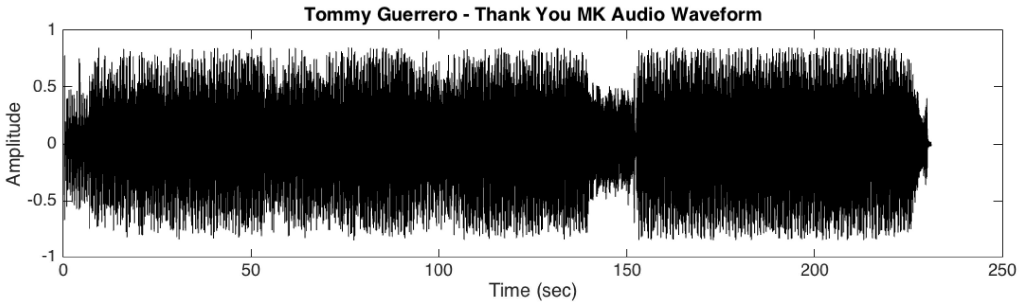

Tinariwen - Ittus Spectral Centroid
Instantaneous (23 ms)
Moving average (2 sec)
Hz
Time (sec)

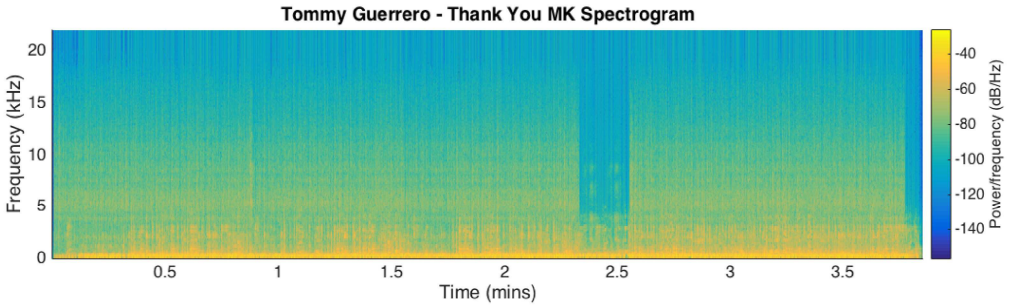

Tinariwen - Ittus Spectral Complexity
Instantaneous (23 ms)
Moving average (2 sec)
N. of peaks
Time (sec)

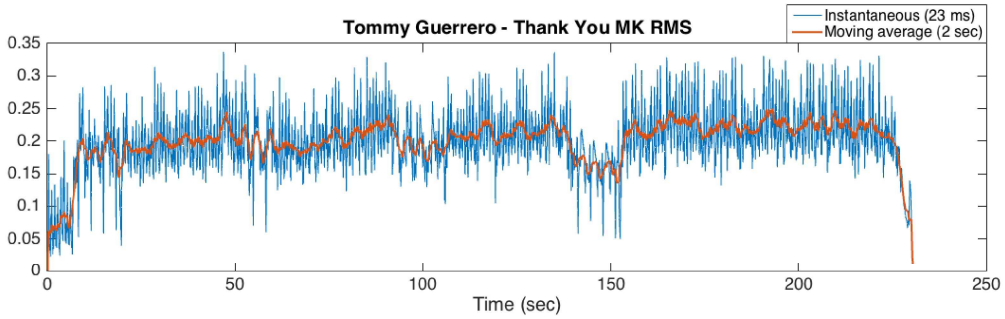

Tinariwen - Ittus Rhythm Strength
2 sec window
Moving average (4 sec)
Time (sec)

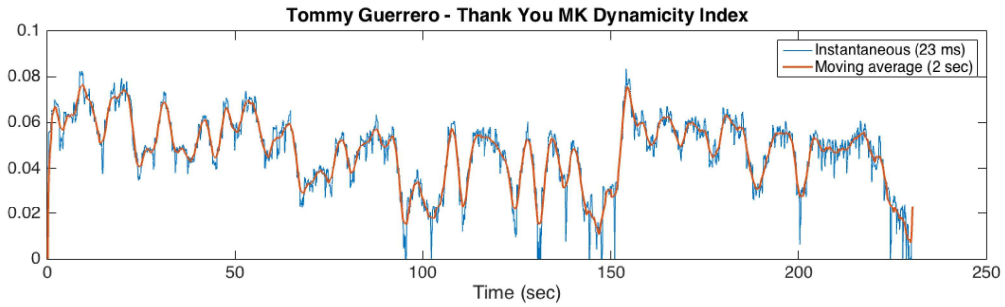

Tinariwen - Ittus Rhythm Density
2 sec window
Moving average (4 sec)
Time (sec)

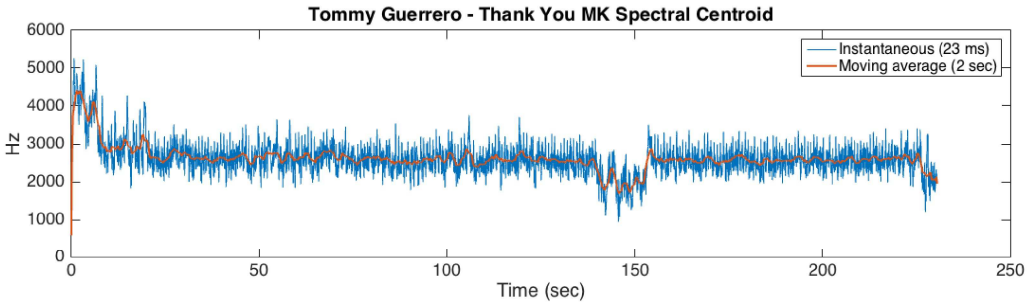
A.10 Tommy Guerrero - Thank You MK

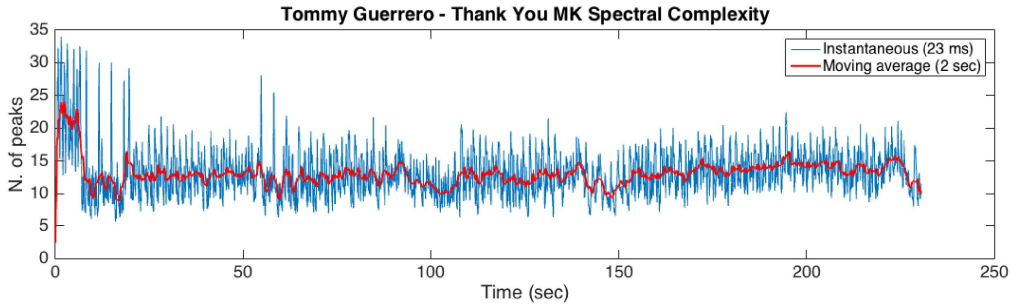


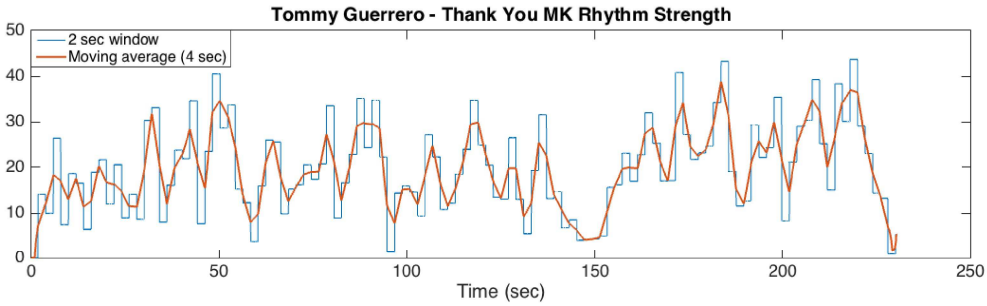


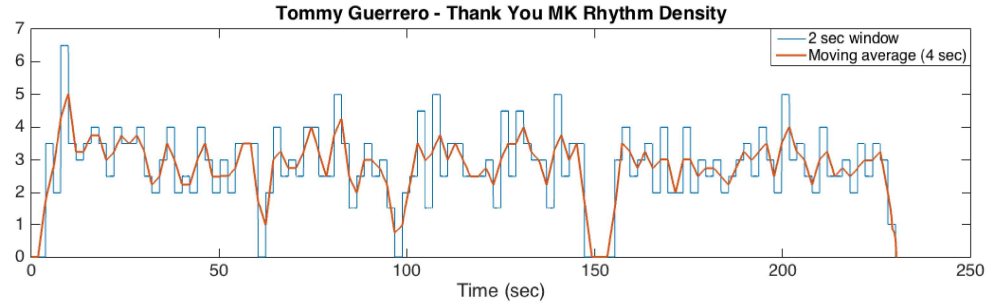




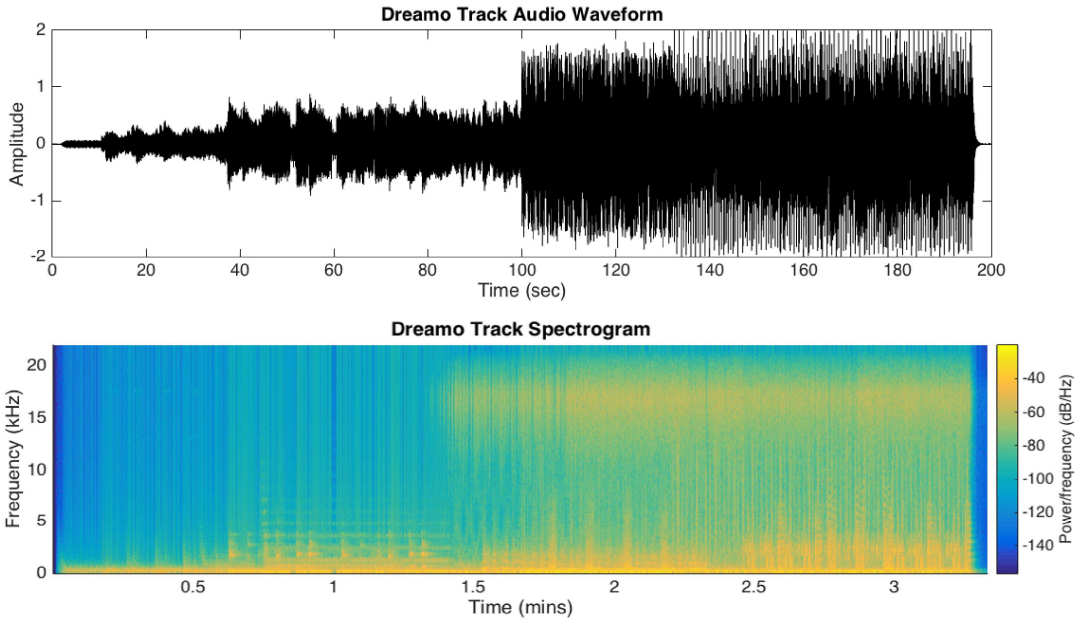

Tommy Guerrero - Thank You MK Spectral Centroid
Instantaneous (23 ms)
Moving average (2 sec)
Hz
Time (sec)

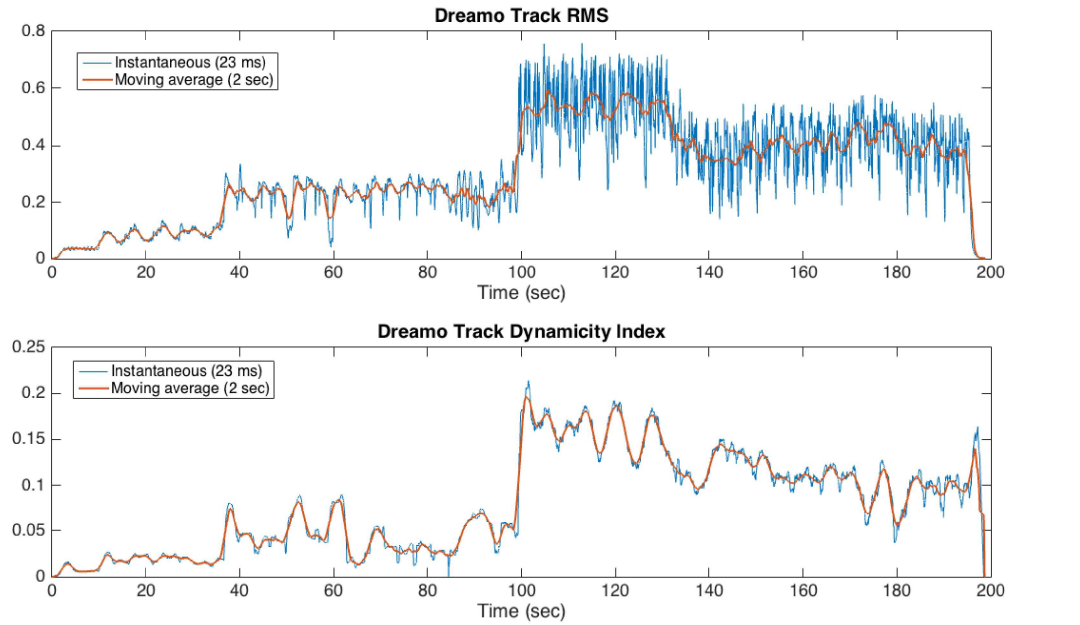

Tommy Guerrero - Thank You MK Spectral Complexity
Instantaneous (23 ms)
Moving average (2 sec)
N. of peaks
Time (sec)

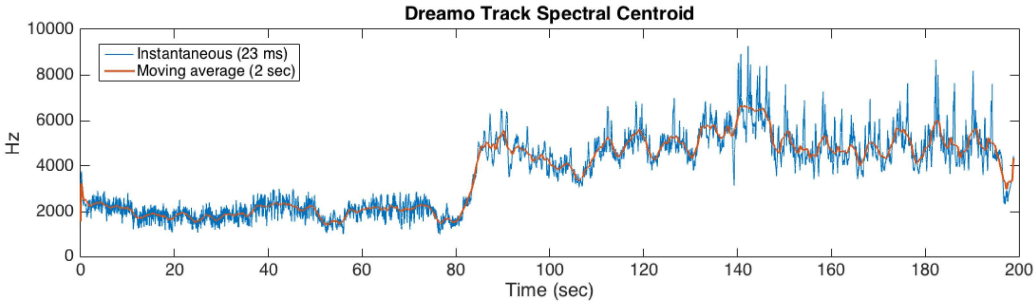

Tommy Guerrero - Thank You MK Rhythm Strength
2 sec window
Moving average (4 sec)
Time (sec)

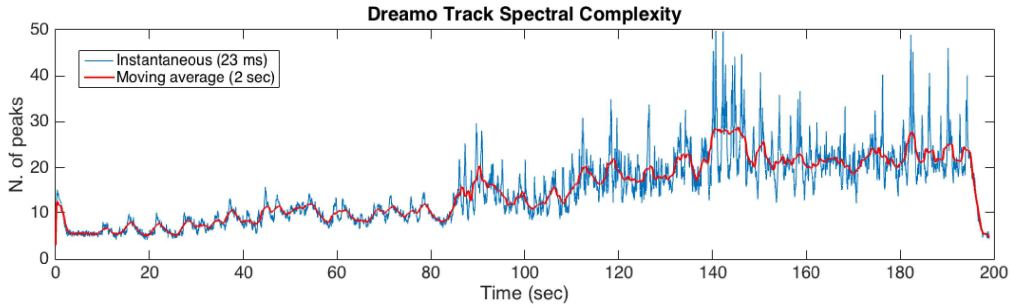

Tommy Guerrero - Thank You MK Rhythm Density
2 sec window
Moving average (4 sec)
Time (sec)

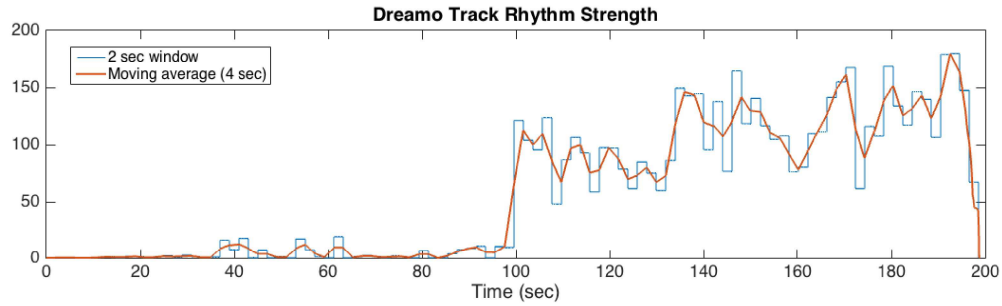
A.11 Dreamo - Track

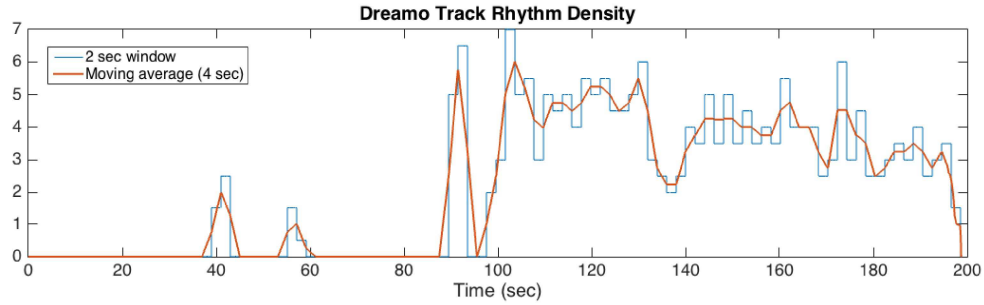





Dreamo Track Spectral Centroid
Instantaneous (23 ms)
Moving average (2 sec)
Hz
Time (sec)


Dreamo Track Spectral Complexity
Instantaneous (23 ms)
Moving average (2 sec)
N. of peaks
Time (sec)


Dreamo Track Rhythm Strength
2 sec window
Moving average (4 sec)
Time (sec)

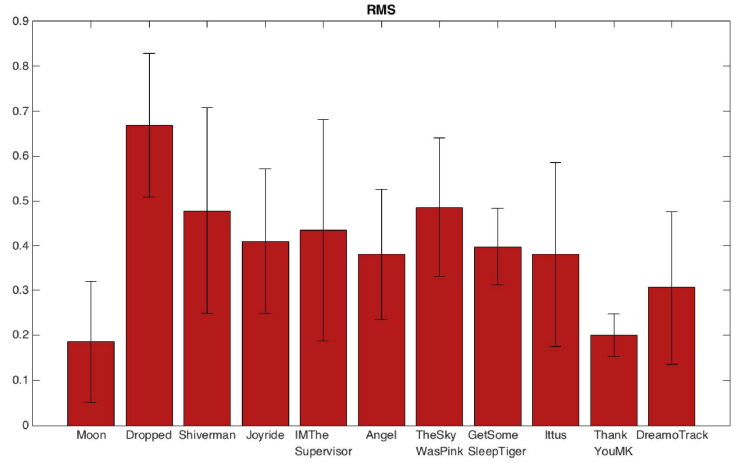

Dreamo Track Rhythm Density
2 sec window
Moving average (4 sec)
Time (sec)

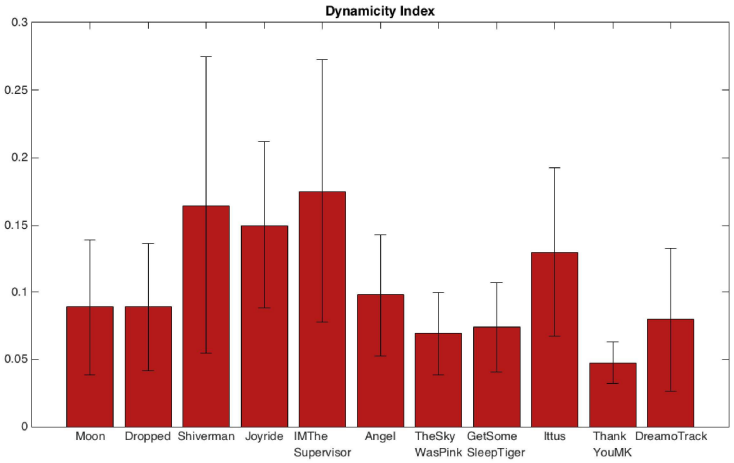
Appendice B

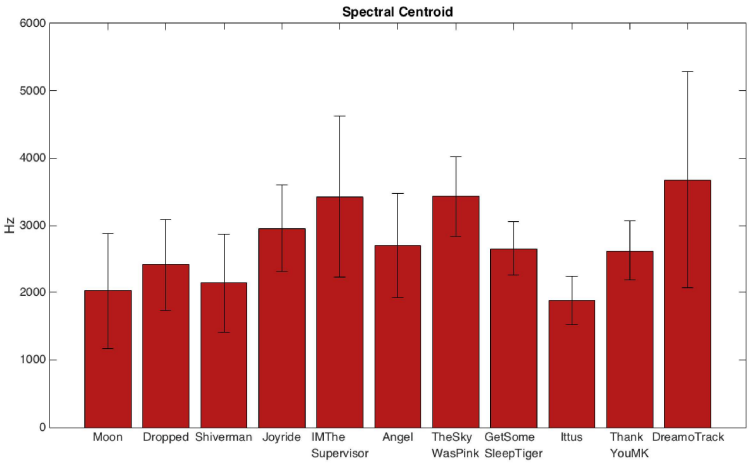
Dati statistici delle features audio

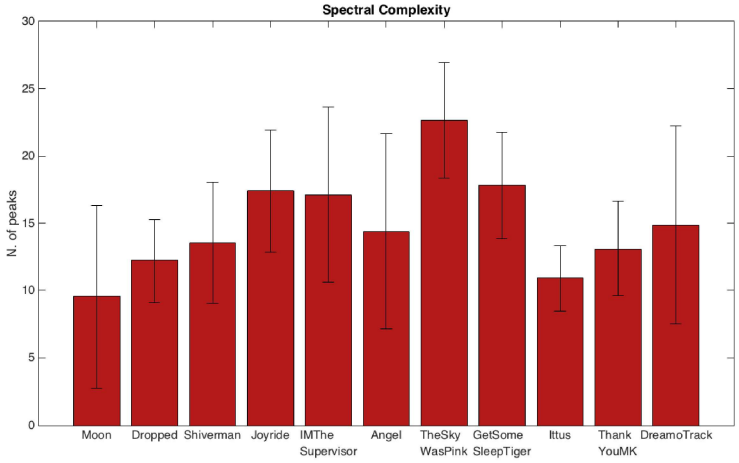
In quest'appendice sono riportati i dati statistici (media e varianza e PDF) di ogni feature calcolati su tutta la durata di ognuno degli 11 brani sopra citati.

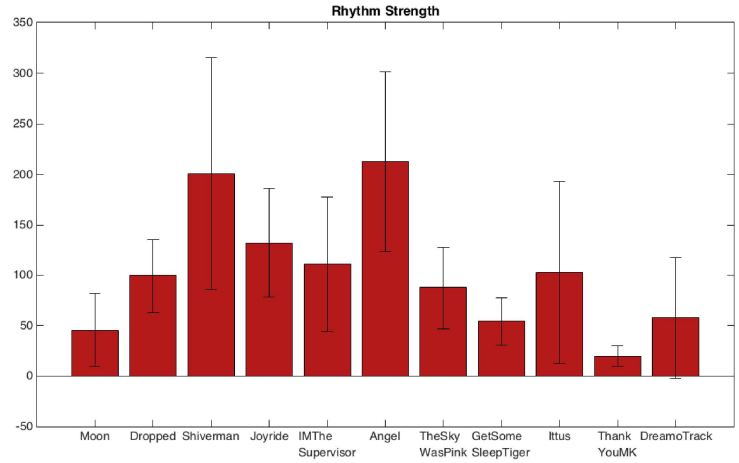
B.1 Grafici di media e deviazione standard

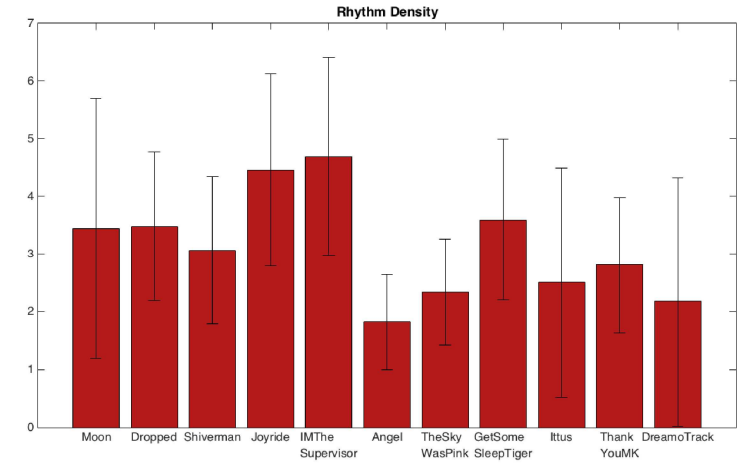







Spectral Complexity
N. of peaks
30
25
20
15
10
5
0
Moon
Dropped
Shiverman
Joyride
IMThe Supervisor
Angel
TheSky WasPink
GetSome SleepTiger
Ittus
Thank YouMK
DreamoTrack


Rhythm Strength
350
300
250
200
150
100
50
0
-50
Moon
Dropped
Shiverman
Joyride
IMThe Supervisor
Angel
TheSky WasPink
GetSome SleepTiger
Ittus
Thank YouMK
DreamoTrack


Rhythm Density
7
6
5
4
3
2
1
0
Moon
Dropped
Shiverman
Joyride
IMThe Supervisor
Angel
TheSky WasPink
GetSome SleepTiger
Ittus
Thank YouMK
DreamoTrack

B.2 Grafici delle PDF

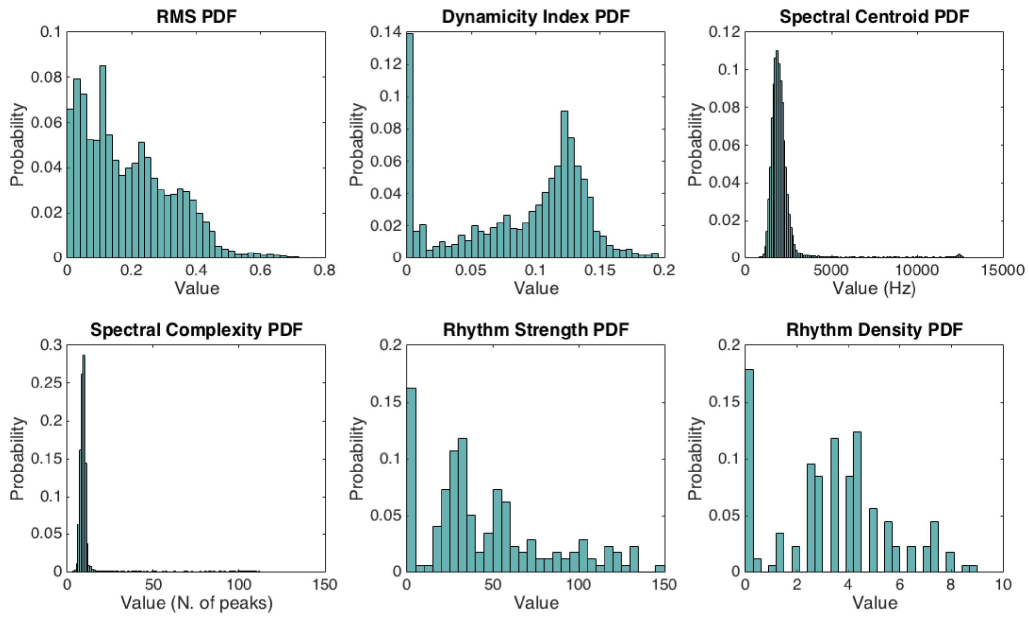


Fig. B2.1: Grafici delle PDF di ogni feature ricavata dal brano "Alva Noto - Moon"

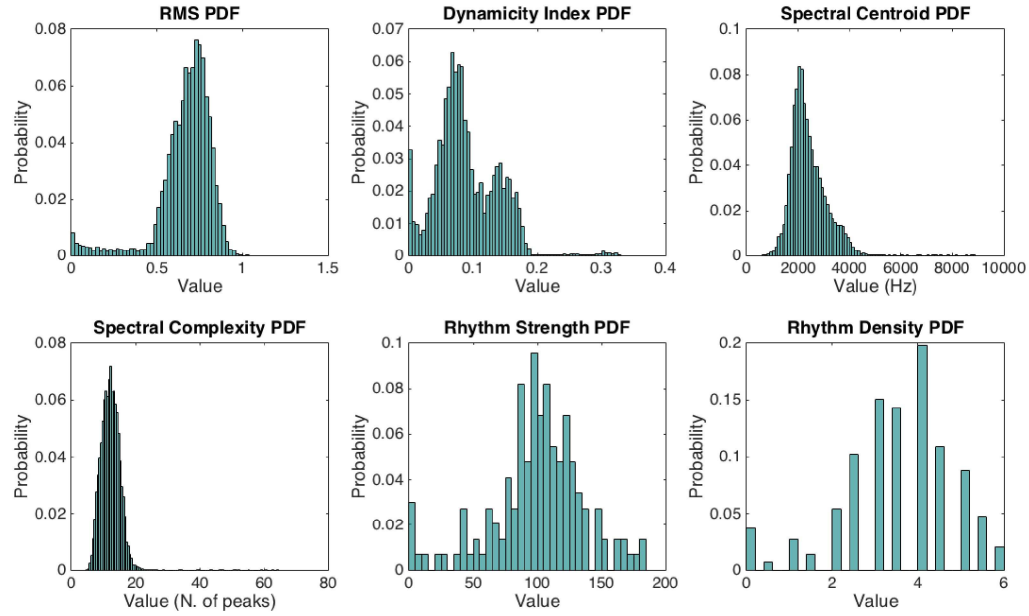


Fig. B2.2: Grafici delle PDF di ogni feature ricavata dal brano "Atoms For Peace - Dropped"

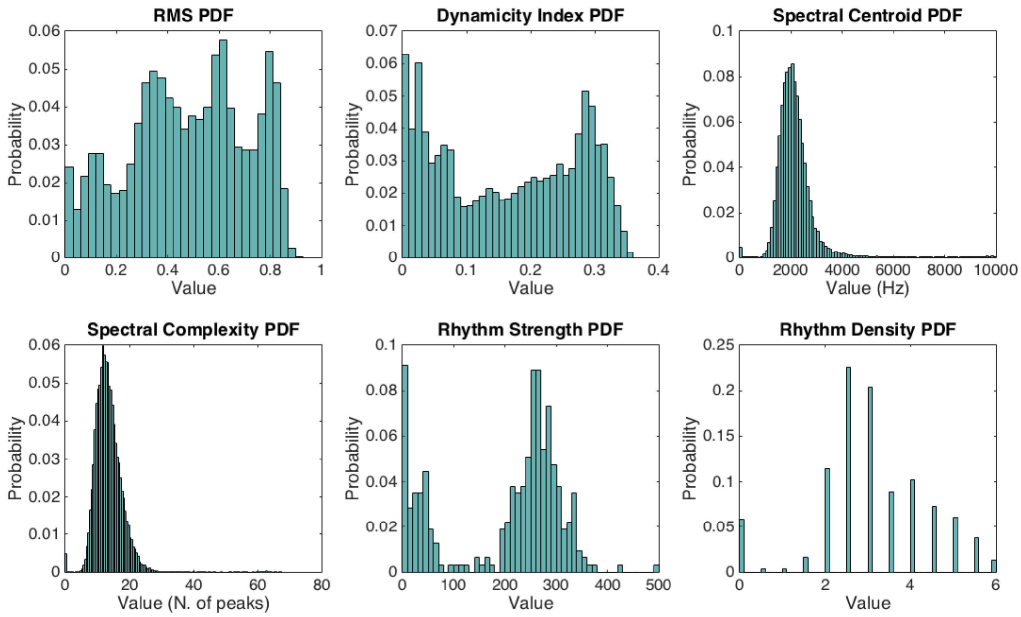


Fig. B2.3: Grafici delle PDF di ogni feature ricavata dal brano "Fat Freddys Drop - Shiverman"

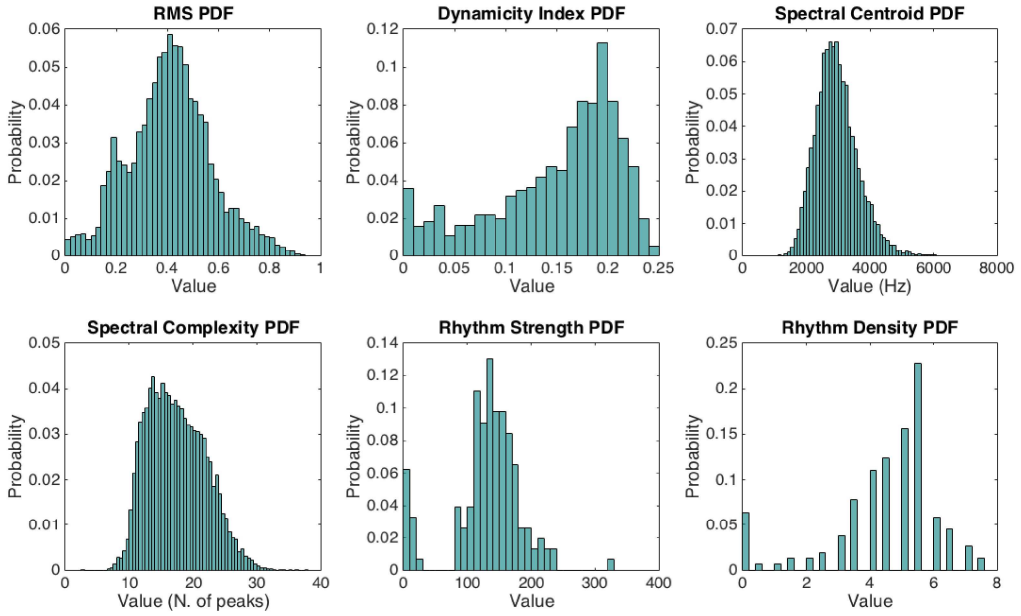


Fig. B2.4: Grafici delle PDF di ogni feature ricavata dal brano "Heliocentric - Joyride"

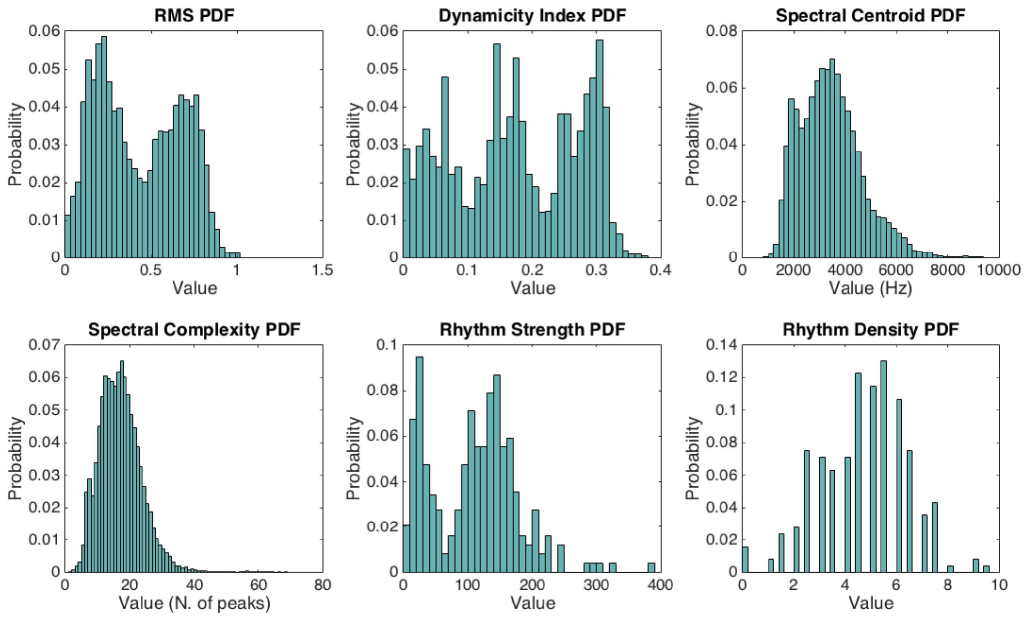


Fig. B2.5: Grafici delle PDF di ogni feature ricavata dal brano "Infected Mushroom - IM The Supervisor"

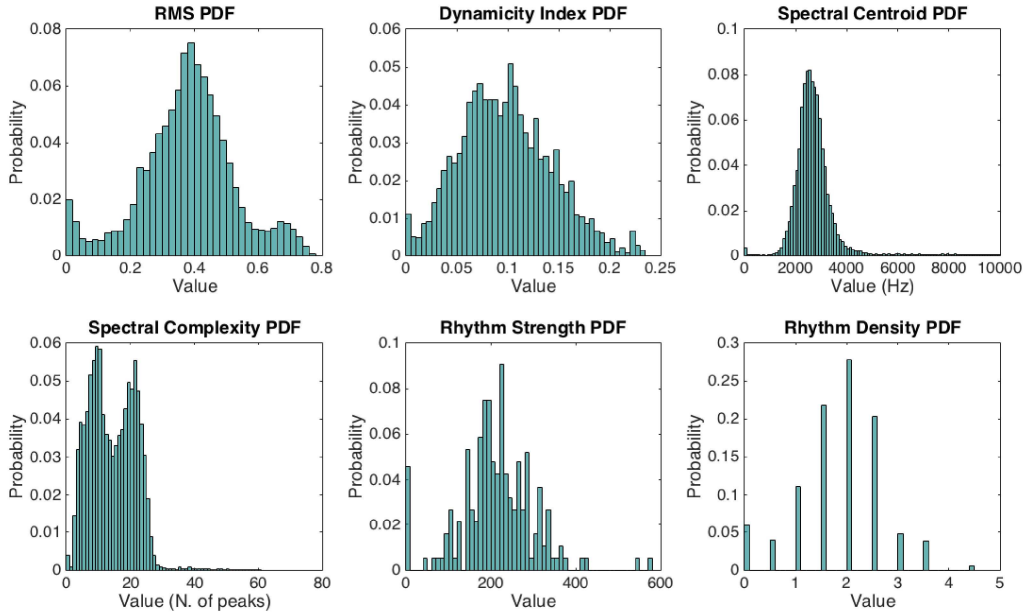


Fig. B2.6: Grafici delle PDF di ogni feature ricavata dal brano "Massive - Attack Angel"

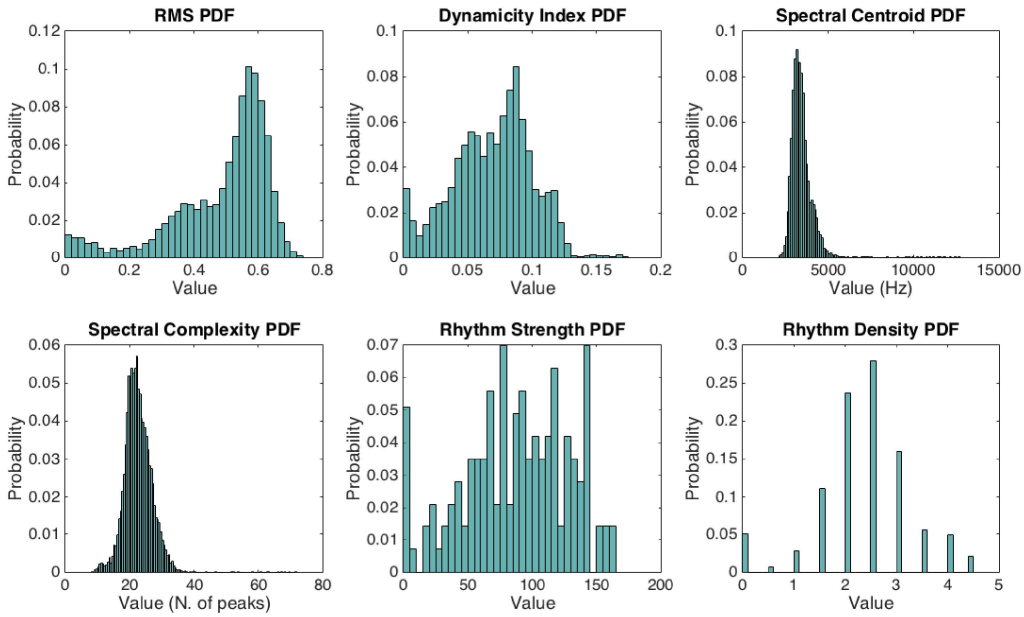


Fig. B2.7: Grafici delle PDF di ogni feature ricavata dal brano "Nathan Fake - The Sky Was Pink"

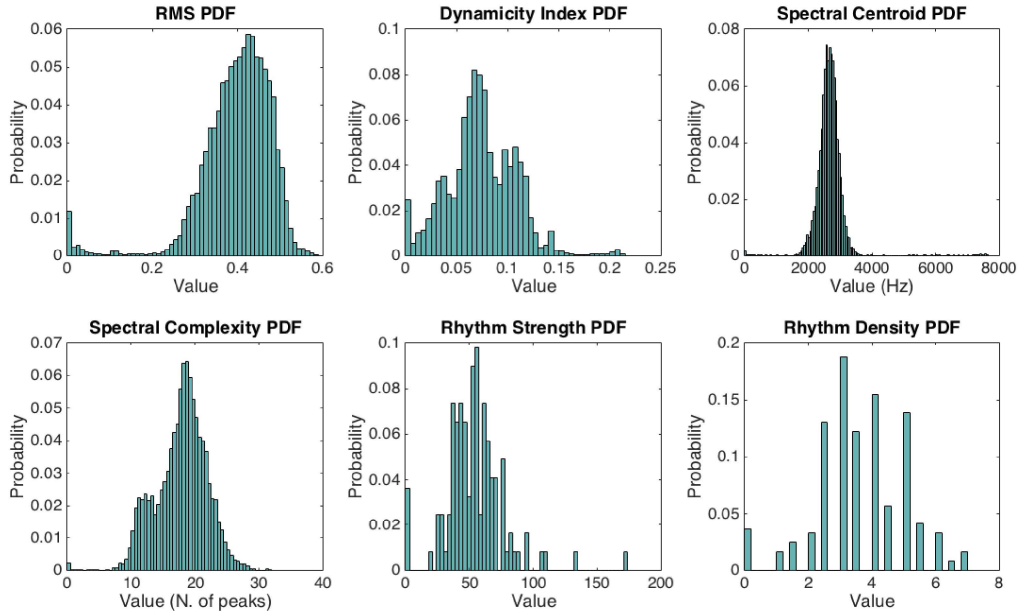


Fig. B2.8: Grafici delle PDF di ogni feature ricavata dal brano "Red Snapper Get Some Sleep Tiger"

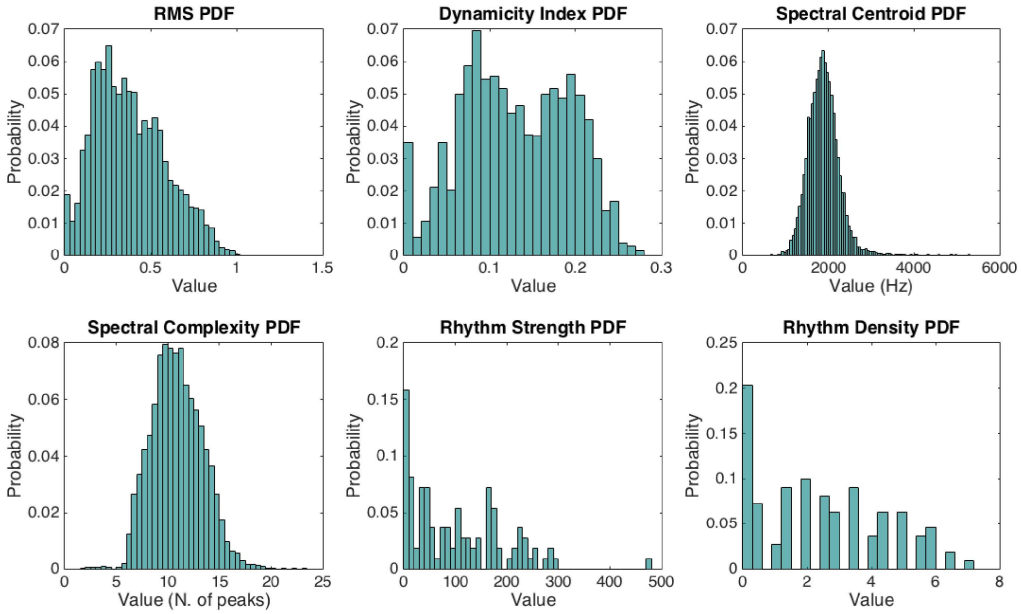


Fig. B2.9: Grafici delle PDF di ogni feature ricavata dal brano "Tinariwen - Ittus"

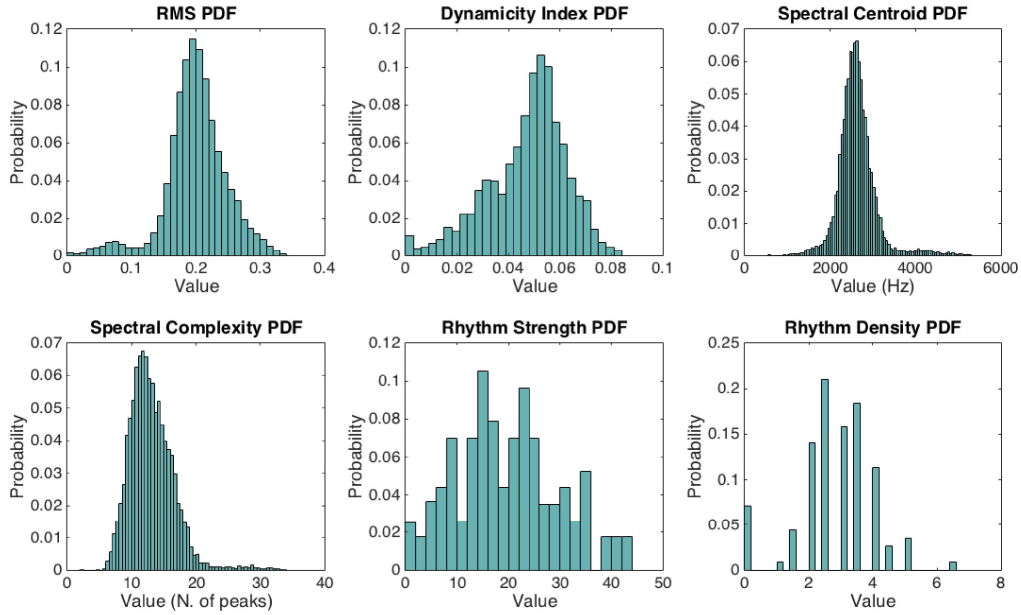


Fig. B2.10: Grafici delle PDF di ogni feature ricavata dal brano "Tommy Guerreo - Thank You MK"

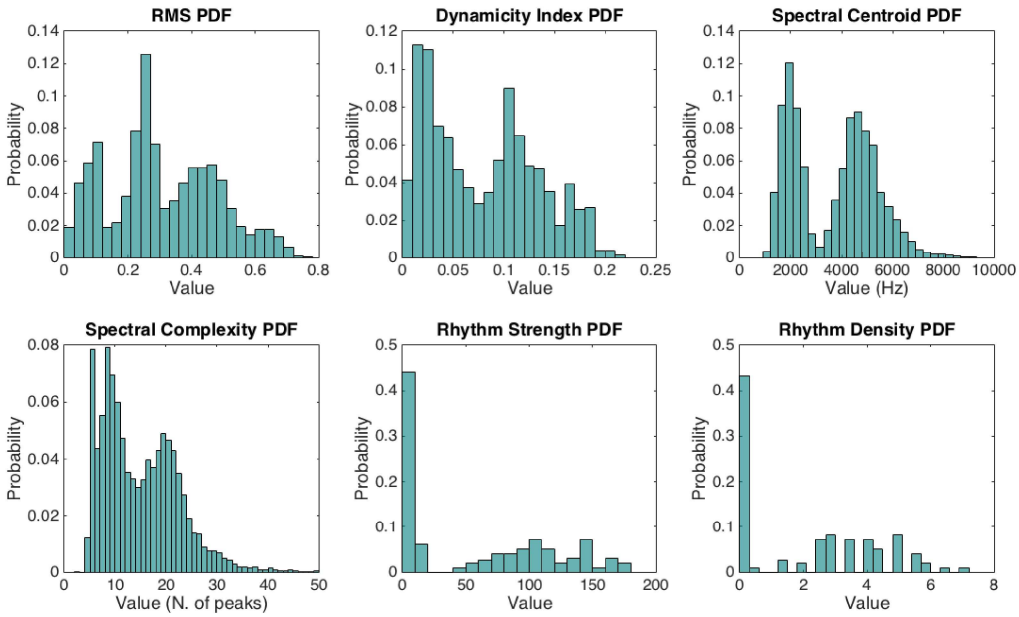


Fig. B2.11: Grafici delle PDF di ogni feature ricavata dal brano "Dreamo Track"

Appendice C

Codice funzioni parametri grafici

La seguente sezione include il codice delle funzioni che hanno il compito di generare i parametri grafici in base ai parametri ottenuti dall'Audio Decisor. Questi metodi sono implementati nella classe padre Scena e sono quindi presenti nel file *class_Scene.pde*.

```
protected void colorFadeTo(Palette p, int seconds, boolean activate)
  {
    if (this.pal.getID()!=p.getID()) //if it's not the same palette
    {
      if (activate || isFading) {
        //if the scene is not already fading
        if (!isFading) //set starting and ending frames
        {
          startFading=frameCount;
          endFading=frameCount+seconds*global_fps;
          targetPal=p;
          isFading=true; //now scene is fading
        } else  //continue fading
        {
          if (frameCount<=endFading)//fading finished?
          {
            //set the colors of the palette
            for (int i=0; i<this.pal.paletteSize(); i++)
            {

              println("... FADING... ");
              this.pal.setColor(lerpColor(this.pal.getColor(i), targetPal.getCol-
or(i), map(frameCount, startFading, endFading, 0, 1)), i);
            }
          } else //finished
          {
            this.setPalette(targetPal); //overwrite palette
            isFading=false;
          }
        }
      }
    }
  }



protected void darkenColors(int seconds, float amount, boolean activate)
  {
    if (activate || changingBrightness) {

      //if the scene is not already fading
      if (!changingBrightness) //set starting and ending frames
      {
        //startFading=frameCount;
        endFading=frameCount+seconds*global_fps;
        changingBrightness=true; //now scene is fading
      } else  //continue fading
      {
        if (frameCount<=endFading)//fading finished?
        {
          //set the colors of the palette
          this.pal.makeDarker(amount/(seconds*global_fps));
        } else //finished
        {
          changingBrightness=false;
        }
      }
    }
  }



protected void lightenColors(int seconds, float amount, boolean activate)
  {
    if (activate || changingBrightness) {
```

```
      //if the scene is not already fading
      if (!changingBrightness) //set starting and ending frames
      {
        //startFading=frameCount;
        endFading=frameCount+seconds*global_fps;
        changingBrightness=true; //now scene is fading
      } else  //continue fading
      {
        if (frameCount<=endFading)//fading finished?
        {
          //set the colors of the palette
          this.pal.makeBrighter(amount/(seconds*global_fps));
        } else //finished
        {
          changingBrightness=false;
        }
      }
    }
  }



protected void saturateColors(int seconds, float amount, boolean activate)
  {
    if (activate || changingSaturation) {

      //if the scene is not already fading
      if (!changingSaturation) //set starting and ending frames
      {
        //startFading=frameCount;
        endFading=frameCount+seconds*global_fps;
        changingSaturation=true; //now scene is fading
      } else  //continue fading
      {
        if (frameCount<=endFading)//fading finished?
        {
          println("SATURATING");

          this.pal.saturate(amount/(seconds*global_fps));
        } else //finished
        {
          changingSaturation=false;
        }
      }
    }
  }



protected void desaturateColors(int seconds, float amount, boolean activate)
  {
    if (activate || changingSaturation) {

      //if the scene is not already fading
      if (!changingSaturation) //set starting and ending frames
      {
        //startFading=frameCount;
        endFading=frameCount+seconds*global_fps;
        changingSaturation=true; //now scene is fading
      } else  //continue fading
      {
        if (frameCount<=endFading)//fading finished?
        {
          //set the colors of the palette
```

```
          this.pal.desaturate(amount/(seconds*global_fps));
        } else //finished
        {
          changingSaturation=false;
        }
      }
    }
  }



protected String choosePaletteFromAudio()
  {
    //TODO: tune this parameter
    if (audio_decisor.getPaletteIndicator()>PALETTE_THRESHOLD) {
      return “warm”;
    } else {
      return “cold”;
    }
  }


protected float chooseElasticityFromAudio()
  {
    float el=0;

    switch((int)audio_decisor.getElasticityIndicator())
    {
      case(0):
      el= 0.4;
      break;

      case(1):
      el= 0.7;
      break;

      case(2):
      el=1;
      break;

      case(3):
      el= 1.3;
      break;

      case(4):
      el= 1.7;
      break;

      case(6):
      el= 2.2;
      break;
    }
    return el;
  }


protected float chooseRotationFromAudio()
  {
    float rot=0;

    switch((int)audio_decisor.getElasticityIndicator())
    {
      case(0):
      rot= 0.1;
      break;
```

```
      case(1):
      rot= 0.09;
      break;

      case(2):
      rot= 0.05;
      break;

      case(3):
      rot= 0.01;
      break;

      case(4):
      rot= 0.005;
      break;

      case(6):
      rot= 0;
      break;
    }
    return rot;
  }


protected float chooseThicknessFromAudio()
  {
    float t=0;

    switch((int)audio_decisor.getTimbreIndicator())
    {
      case(0):
      t = 40;
      break;

      case(1):
      t = 30;
      break;

      case(2):
      t = 17;
      break;

      case(3):
      t = 11;
      break;

      case(4):
      t = 7;
      break;

      case(6):
      t = 3;
      break;
    }
    return t;
  }



protected float chooseVibrationFromAudio()
  {
    float vib=0;

    switch((int)audio_decisor.getVibrationIndicator())
```

```
    {
      case(0):
      vib = 1;
      break;

      case(1):
      vib = 1.2;
      break;

      case(2):
      vib = 1.5;
      break;

      case(3):
      vib = 1.8;
      break;

      case(4):
      vib = 2;
      break;

      case(6):
      vib= 3.8;
      break;
    }
    return vib;
  }



protected float chooseVibrationRangeFromAudio()
  {
    float range=0;

    if(audio_decisor.getFeaturesVector()[0]<0.05) {range=3/0.05;}
    else range=3/audio_decisor.getFeaturesVector()[0];

    return range;

  }


protected float chooseResolutionFromAudio()
  {
    float res=0;

           if(audio_decisor.getStatusVector()[3]==0 || audio_decisor.getStatusVec-
       tor()[3]==-1)
    {
      if(audio_decisor.getFeaturesVector()[3]<3 || audio_decisor.getFeaturesVector()
[0]<0.01 ) res=2; //basic shape when silence or very smooth sound
      else if(audio_decisor.getFeaturesVector()[3]<9) res=5; //basic shape for
smooth sounds
      else res=7;
    }
    else if(audio_decisor.getStatusVector()[3]==1)
    {
      res=11;
    }
    else if (audio_decisor.getStatusVector()[3]==3)
    {
      res= audio_decisor.getFeaturesVector()[3];
    }
    return res;
  }
```

Appendice D

Andamento temporale delle features acustiche

La seguente appendice presenta per intero il codice presente nei files *class_scene_CrazyL.pde* e *class_particle_CrazyLines.pde*, responsabile della visualizzazione dell'esempio presentato nel paragrafo 5.3. è inoltre presente il codice di c*lass_scene_perlinNoise.pde* e *class_particle_NoiseDot.pde*, che generano l'effetto grafico presentato nella sezione 5.4.

D.1 class_scene_CrazyL.pde

```
class CrazyL extends Scene

{

  CrazyLines C;

  public void init()
  {
    pal.initColors();
    C = new CrazyLines(5);
    C.setPalette(pal);
    C.disablePhysics();
    C.setPosition(new Vector2d(width/2, height/2, false));
    addParticle(C);

    Background bk = new Background();
    setBackground(bk);
    enableBackground();

  }

  //trace and update methods
  public void update()
  {

    for(int i = 0; i < particlesNumber; i++)
    {
      particlesList[i].updatePhysics();
      particlesList[i].setPalette(this.pal);
      particlesList[i].update();
    }


    //**** COLOR CONTROLS
    colorFadeTo(new        Palette(choosePaletteFromAudio()),2,audio_decisor.getPal-
etteChange()); //choose warm or cold palette
    darkenColors(3,30f, (audio_decisor.getCentroidChangeDir()==-1)); //control color
brightness
    lightenColors(3,50f, (audio_decisor.getCentroidChangeDir()==1)); //control color
brightness
    desaturateColors(3,20f,(audio_decisor.getComplexityChangeDir()==-1)); //control
color saturation
    saturateColors(3,60f,(audio_decisor.getComplexityChangeDir()==1)); //control
color saturation
    colorFadeTo(new Palette(choosePaletteFromAudio()),2,(audio_decisor.getChang-
esNumber()>2)); //follow big changes

    //**** MOTION CONTROLS
    C.setElasticityCoeff(chooseElasticityFromAudio());
    C.setVibrationFreq(chooseVibrationFromAudio());
    C.setRotationCoeff(chooseRotationFromAudio());

    //**** SHAPE CONTROLS
    C.setFormResolution((int)chooseResolutionFromAudio());
    C.setThickness(chooseThicknessFromAudio());
    C.setVibrationRange(chooseVibrationRangeFromAudio());

   }
```

```
}
```

D.2 class_particle_CrazyLines.pde

```
import java.util.List;
import java.util.ArrayList;

class CrazyLines extends Particle
{

  int formResolution;
  int targetResolution;
  boolean morphing=false;

  float initRadius = 50;

  //POINT ARRAYS
  List<Float> x = new ArrayList<Float>();
  List<Float> y = new ArrayList<Float>();

  //COLOR
  color myColor;
  int colorIDX;

  //ROTATION
  float count = 0;

  float distortionCoeff;
  float elasticityCoeff=1;
  float rotationCoeff=0.009;

  //STROKE WEIGTH CONTROL
  float vibrationFreq=1;
  float vibrationRange=10;
  float weightSeed=10;

  //stroke weigth vibration controls
  float startingWeight;
  float targetWeight;
  float weight;
  float oldWeight;


  public CrazyLines()
  {
   formResolution = 5; //DEFAULT RESOLUTION
   targetResolution=formResolution;
  }


  public CrazyLines(int res)
  {
   formResolution=res;
   targetResolution=formResolution;
  }


  public void init()
  {
   setColorIndex((int)random(0,5));
```

```
   float angle = radians(360/float(formResolution));

   for (int i=0; i<formResolution; i++)
   {
     x.add(i, (cos(angle*i) * initRadius));
     y.add(i, (sin(angle*i) * initRadius));
   }


   startingWeight=random(weightSeed-vibrationRange/2,weightSeed+vibrationRange/2);
   targetWeight=random(weightSeed-vibrationRange/2,weightSeed+vibrationRange/2);

   weight=startingWeight;
   oldWeight=weight;

  }


  public void update()
  {
    //**** COLOR
    myColor = pal.getColor(getColorIndex());

    //**** PARAMETERS FOR DIRECT INFLUENCE
    setParameter(0,audio_decisor.getInstantFeatures()[0]);
    setParameter(1,pow(5,audio_decisor.getFeaturesVector()[2]/2500));
    setParameter(2,audio_decisor.getFeaturesVector()[0]);
    setParameter(3,audio_decisor.getFeaturesVector()[3]);
    setParameter(4,audio_decisor.getFeaturesVector()[2]);

    //**** DIRECT INFLUENCE of TIMBRE on colors SATURATION AND BRIGHTNESS
    pal.influenceColors(0,mapForSaturation(getParameter(3),0,40),mapForBright-
ness(getParameter(4),0,6000));

    //**** SHAPE RESOLUTION
    changeShape();

    //**** STROKE WEIGHT
    changeStrokeWeight();

  }

  public void trace()
  {
    noFill();

    //**** STROKE WEIGHT VIBRATION ****
    stroke(myColor);

    strokeWeight(weight);

    //**** SCALE PULSATION ****
    scale(0.5+getParameter(0)*elasticityCoeff); //DIRECT INFLUENCE of RMS (instan-
taneous value) on SCALE

    //**** SHAPE GENERATION ****
    float angle = radians(360/float(formResolution));

    beginShape();

    curveVertex(x.get(formResolution-1)*getParameter(1), y.get(formResolu-
tion-1)*getParameter(1)); //start controlpoint

    //only these points are drawn
```

```
    for (int i=0; i<formResolution; i++)
    {

        x.add(i, (cos(angle*i) * width/10));
        y.add(i, ( sin(angle*i) * width/10));

        count = count + radians(getParameter(2)*rotationCoeff); //DIRECT INFLUENCE
of RMS (average) on ROTATION
        rotate(count);

        curveVertex(x.get(formResolution-1)*getParameter(2), y.get(formResolu-
tion-1)*getParameter(2));
        curveVertex(x.get(i)*getParameter(1), y.get(i)*getParameter(1));

    }

    curveVertex(x.get(1)*getParameter(1), y.get(1)*getParameter(1)); //end control-
point

    endShape();

  }



  private void changeShape()
  {

    if(targetResolution==formResolution)
    {
      morphing=false;
    }

    else if((targetResolution-formResolution)>0)
    {
      formResolution++;
      morphing=true;

      float angle = radians(360/float(formResolution));

      for (int i=0; i<formResolution; i++)
      {
        x.add(i, (cos(angle*i) * initRadius));
        y.add(i, (sin(angle*i) * initRadius));
       }
    }

    else if((targetResolution-formResolution)<0)
    {
      formResolution--;
      morphing=true;

      float angle = radians(360/float(formResolution));
      for (int i=0; i<formResolution; i++)
      {
        x.add(i, (cos(angle*i) * initRadius));
        y.add(i, (sin(angle*i) * initRadius));
       }
    }
  }


  private void changeStrokeWeight()
  {
```

```
    if((oldWeight-targetWeight)*(weight-targetWeight)<0) //zero crossing -> target
reached!
    {
      startingWeight=weight;
      oldWeight=weight;
      targetWeight=random(weightSeed-vibrationRange/2,weightSeed+vibrationRange/2);
    }

    else if((weight-targetWeight)<0)
    {
      oldWeight=weight;
      weight+=(vibrationFreq);

    }

    else if((weight-targetWeight)>0)
    {
      oldWeight=weight;
      weight-=(vibrationFreq);
    }

  }


  public void setFormResolution(int res)
  {
    if(!morphing)targetResolution=res; //if not morphing, assign new target
  }

  public void setElasticityCoeff(float coeff)
  {
    elasticityCoeff=coeff;
  }

 public void setVibrationFreq(float f)
 {
   vibrationFreq=f;
 }

 public void setThickness(float t)
 {
   weightSeed=t;
 }

 public void setVibrationRange(float r)
 {
   vibrationRange=r;
 }

 public void setRotationCoeff(float rot)
 {
   rotationCoeff=rot;
 }



}
```

D.3 class_scene_perlinNoise.pde

```
class ScenePerlinNoise extends Scene
{


  void init()
  {
    pal.initColors();
    final int row = 10;
    final int column = 11;
    for(int i = 0; i < column; i++)
    {
      for(int j = 0; j < row; j++)
      {
        int x = round(width/column*(i+1));
        int y = round(height/row*(j+1));
        NoiseDot temp = new NoiseDot();
        temp.setPalette(pal);
        temp.setPosition(new Vector2d(x, y, false));
        addParticle(temp);
      }
    }
    setBackground(new Background());
    enableBackground();
  }

  public void update()
  {
    colorFadeTo(new Palette(choosePaletteFromAudio()),2,audio_decisor.getPal-
etteChange()); //choose warm or cold palette

    for(int i = 0; i < particlesNumber; i++)
    {
      particlesList[i].updatePhysics();
      particlesList[i].setPalette(this.pal);
      particlesList[i].setParameter(2,1.5*chooseThicknessFromAudio());
      particlesList[i].setParameter(3,1200/chooseVibrationFromAudio());
      particlesList[i].setParameter(4,200*chooseElasticityFromAudio());
      particlesList[i].update();
    }
  }
}
```

D.4 class_particle_NoiseDot.pde

```
class NoiseDot extends Particle
{
  float speed;
  float dotW;
  float dotH;
  float noiseScale;
  float noiseStrength;
  float angle;
  float nCrossedX, nCrossedY;
  int indexShifting;

  void init()
  {
    speed = 10;
    dotW=9;
    dotH=9;
    noiseStrength = 1;
    noiseScale = 300;
    nCrossedX = 0;
    nCrossedY = 0;
    indexShifting = 0;
  }

  void update()
  {
    //**** PARAMETERS FOR DIRECT INFLUENCE
    setParameter(0,audio_decisor.getInstantFeatures()[0]); //RMS istantaneo
    setParameter(1,audio_decisor.getFeaturesVector()[0]); //RMS mediato (2 sec)

    //**** PARAMETERS FROM SCENE DECISIONS
    dotW=getParameter(2);
    dotH=getParameter(2);
    noiseScale=getParameter(3);
    noiseStrength=getParameter(4);


    if ( frameCount % 4 == 0 && indexShifting < getPalette().COLOR_NUM && round(ran-
dom(1)) == 1 )     indexShifting++;
    if ( indexShifting >= getPalette().COLOR_NUM)         indexShifting = 0;

    angle = noise((getPosition().getX() + nCrossedX*width)/noiseScale, (getPosi-
tion().getY()+nCrossedY*height)/ noiseScale) * noiseStrength;

    getSpeed().setDirection(angle);


    getSpeed().setModulus(speed*getParameter(0));

    pal.influenceColors(0,mapForSaturation(getParameter(1),0,1),0);

    keepInsideTheScreen();

  }

  public float getAngle()
  {
```

```
    return angle;
  }

  void trace()
  {

    noStroke();
    fill(pal.getColor(indexShifting));
    ellipse(-5, -5, dotW, dotH);

  }

  void keepInsideTheScreen()
  {
     if(getPosition().getX() > width)
    {
      getPosition().setX(getPosition().getX() - width);
      nCrossedX++;
    }
    if(getPosition().getY() > height)
    {
      getPosition().setY(getPosition().getY() - height);
      nCrossedY++;
    }
    if(getPosition().getX() < 0)
    {
      getPosition().setX(getPosition().getX() + width);
      nCrossedX--;
    }
    if(getPosition().getY() < 0)
    {
      getPosition().setY(getPosition().getY() + height);
      nCrossedY--;
    }
  }


}
```