Artist to
Business to Business
to Consumer
Audio Branding System

www.abcdj.eu

# D4.8 Final Research Report for Sound Design and Audio Player

Report on research and development of specific features for the audio player

| | |
|---|---|
| Project Reference | 688122 — ABC_DJ — H2020-ICT-2015 |
| Deliverable/WP/Task | D4.8 WP4 T4.3 |
| Delivery Date | 31/12/2018 |
| Main Author(s) | Diemo Schwarz, Ircam, schwarz@ircam.fr |
| Co-Author(s) | Dominique Fourer, Ircam, dominique.fourer@univ-evry.fr |
| Quality Assurance | Steffen Lepa, TUB, steffen.lepa@tu-berlin.de<br>Alessandro Canepa, PIAC, alessandro.canepa@piacenza1733.it |
| Filename | D4.8_ABC_DJ_Final_Research_Report_for_Sound_Design_and_Audio_Player.pdf |
| Publication Level | PU |

**ABC_DJ - Artist-to-Business-to-Business-to-Consumer Audio Branding System**
contact: www.abcdj.eu
email: info@abcdj.eu

**Project Information**

| | |
|---|---|
| Full project title | ABC_DJ — Artist-to-Business-to-Business-to-Consumer Audio Branding System |
| Project Coordinator | Stefan Weinzierl / TU Berlin |
| Project ID | 688122 — ABC DJ — H2020-ICT-2015 |

**Acknowledgements**

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 688122.

# Table of Contents



# History

| Version | Name | Date | Remark |
|---|---|---|---|
| V0.1 | Diemo Schwarz | 2018-12-19 | Update from D4.6: move cue regions to D4.7 |
| V0.2 | Diemo Schwarz | 2018-12-20 | Update *unmixdb* |
| V0.3 | Diemo Schwarz | 2018-12-21 | Update mixing module data formats and scheduling |
| V0.4 | Diemo Schwarz | 2018-12-22 | Integrate first revisions |
| V0.5 | Diemo Schwarz | 2018-12-27 | Integrate second revisions |
| V1.0 | Diemo Schwarz | 2018-12-30 | Submitted to EC |

# Glossary

| Acronym/Abbreviation | Full Name/Description |
|---|---|
| ABC_DJ | Artist-to-Business-to-Business-to-Consumer audio branding system |
| DTW | dynamic time warping |
| ISP | in-store player |
| JSON | JavaScript Object Notation |
| PLG | playlist generator |
| PoS | point of sale |
| RMS | root-mean-square |
| SW | software |
| xfade | cross-fade |

# Executive Summary

This deliverable describes the work on Task 4.3 *Algorithms for sound design and feature developments for audio player*. The audio player runs on the in-store player (ISP) and takes care of rendering the music playlists via beat-synchronous automatic DJ mixing, taking advantage of the rich musical content description extracted in T4.2 (beat markers, structural segmentation into intro and outro, musical and sound content classification).

The deliverable covers prototypes and final results on: (1) automatic beat-synchronous mixing by beat alignment and time stretching – we developed an algorithm for beat alignment and scheduling of time-stretched tracks; (2) compensation of play duration changes introduced by time stretching – in order to make the playlist generator independent of beat mixing, we chose to readjust the tempo of played tracks such that their stretched duration is the same as their original duration; (3) prospective research on the extraction of data from DJ mixes – to alleviate the lack of extensive ground truth databases of DJ mixing practices, we propose steps towards extracting this data from existing mixes by alignment and unmixing of the tracks in a mix. We also show how these methods can be evaluated even without labelled test data, and propose an open dataset for further research; (4) a description of the software player module, a GUI-less application to run on the ISP that performs streaming of tracks from disk and beat-synchronous mixing.

The estimation of cue points where tracks should cross-fade is now described in *D4.7 Final Research Report on Auto-Tagging of Music*

# 1 Introduction

This deliverable describes the work on Task 4.3 *Algorithms for sound design and feature developments for audio player*. The audio player runs on the in-store player and takes care of rendering the music playlists via beat-synchronous automatic DJ mixing, taking advantage of the rich musical content description extracted in T4.2 (beat markers, structural segmentation into intro and outro, musical and sound content classification).
The work is based on 3 sets of example data provided by HearDis!:

1. example mixes: two DJ mixes in the format of the authoring software *MixMeister* and *Ableton Live*. They comprise the individual source tracks and the arrangement and mix curves for volume and EQ.
2. cue point examples: a set of 30 example tracks in mp3 format, each in two versions:
    1. the full-length track
    2. the track shortened according to human-decided cue-in and cue-out regions with fades
3. tagged MLM examples: 4 sets of tracks (low, mid, high tempo, classical) with complete annotations, each set of a certain tempo group

The deliverable mainly covers research and development results on:

- automatic beat-synchronous mixing by beat alignment within the cue point regions and time stretching. This is based on algorithms for play duration compensation, and beat alignment.
- prospective research on the extraction of data from DJ mixes
- a description of the architecture and i/o formats of the software player

# 2 Automatic Beat-Synchronous Mixing

The parts of the development of T4.3 described in this section concern the automatic DJ mixing function of the in-store player (ISP) that is meant to simulate a broadcast or lounge DJ mix where tracks are mixed seamlessly with matching tempi and synchronised beats during the cross-fade phase. This is based on the tempo and beat detection of T4.2 *Algorithms for extraction, automatic classification and indexing of music*.
For beat-synchronicity, time-stretching will be carried out (without altering the pitch of the tracks). For this, three options are conceivable:

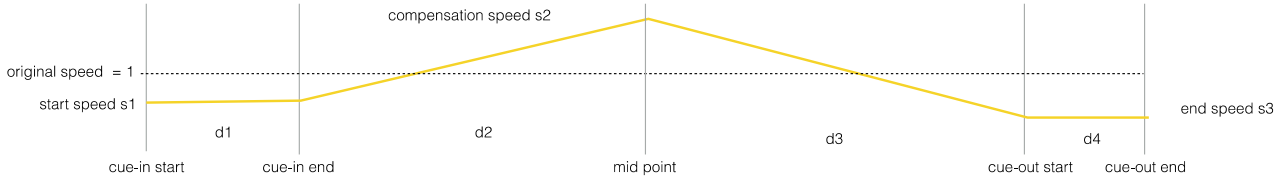
1. Only the first song is slowed down/sped up before the transition to fit the following one.
2. Both songs are slowed down/sped up to meet at an average BPM. After the transition the song returns to original BPM.
3. Both songs are slowed down/sped up to meet at an average BPM. After the transition, the song is slowed down/sped up in the opposite direction to reverse the change in duration.

Option 3, which was chosen for realization, has the advantage of not requiring the playlist generator to be aware of the change in duration due to beat-synchronous mixing (because the PLG can be used to generate mixes with a certain song playing at a specific predefined time point, e.g. for shop closing time, noon, etc.).
As future work, in case option 3 introduces speed changes (accelerations/slowdowns) that are too noticeable[1], an alternative approach would be to have the PLG transmit to the ISP mixing module which song needs to start at a specific predefined time point, so that that this requirement can be met by adapting the speeds of tracks in the playlist up to that point (which don't need to start at a specific time point), thus distributing speed changes over a longer period.

The quality of the algorithms and different settings of parameters generating beat-synchronous mixes ought to be evaluated by qualitative listening tests performed by experts and laypeople at a future point in time. Such experts can be the HearDis! music specialists who are able to judge the success of the algorithms in terms of achieving human DJ-like mixing quality.

## 2.1 Duration Compensation

In order to realise option 3 from above, we formalised the compensation of play duration due to beat-synchronous mixing. We defined a 4-phase time model shown in the following figure:




*Depiction of 4-phase time model for duration compensation.*

The speed curve (yellow line) has 4 linear segments with durations *d1*...*d4*, where the cross-fade segments *d1* and *d4* are of constant speeds *s1* and *s3* determined by the previous/next track, their duration being determined by the cue-in and cue-out region. Segments *d2* and

1 What "too noticeable" means would have to be defined by listening tests.

*d3* serve to compensate the duration changes introduced thereby such that the sum of the durations *after applying the speed curve* add up to the original duration of the track, i.e. $d1' + d2' + d3' + d4' = duration$. Here, $dn'$ is the notation for stretched duration of a segment *n* after applying its tempo curve.

The only free parameter is thus the compensation speed *s2* and the time point where it is reached. We choose the time point in the middle, i.e. $d2 = d3$. As it is easy to see, duration compensation is reached when the area under the yellow speed curve is equal to that under the dotted original speed curve, or, equivalently, when the area normalised by target track duration is equal to 1. This normalised area is given by the sum of the segment areas:

$$area = (a1 + a2 + a3 + a4)/dtarget$$

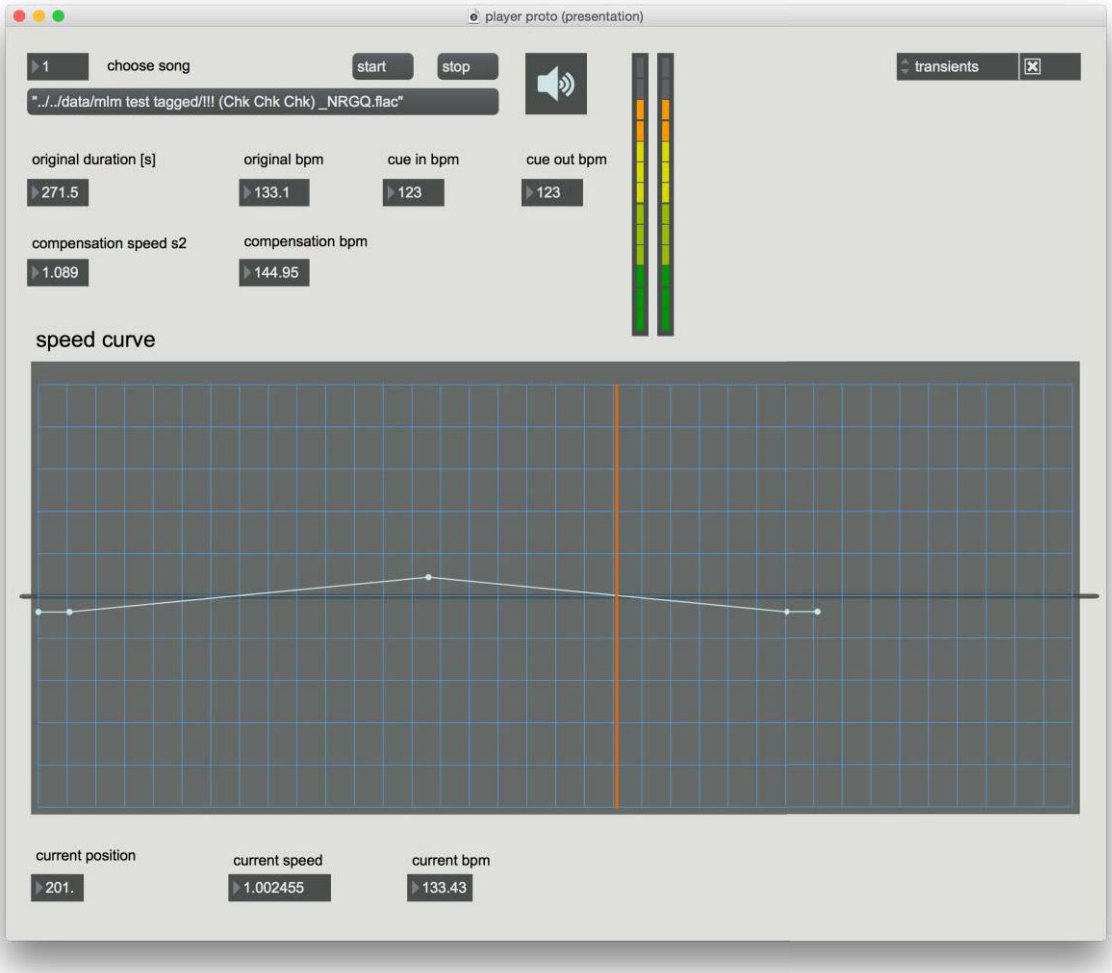
with

$$a1 = s1 * d1'$$
$$a2 = (s1 + s2)/2 * d2' = (s2 - s1)/2 * d2' + s1 * d2'$$
$$a3 = (s2 + s3)/2 * d3' = (s3 - s2)/2 * d3' + s2 * d3'$$
$$a4 = s3 * d4'$$

Solving for s2 yields

$$s_2 = \frac{d_{target} - s_1 d'_1 - \frac{1}{2} s_1 d'_2 - \frac{1}{2} s_3 d'_3 - s_3 d'_4}{\frac{1}{2}(d'_2 + d'_3)}$$

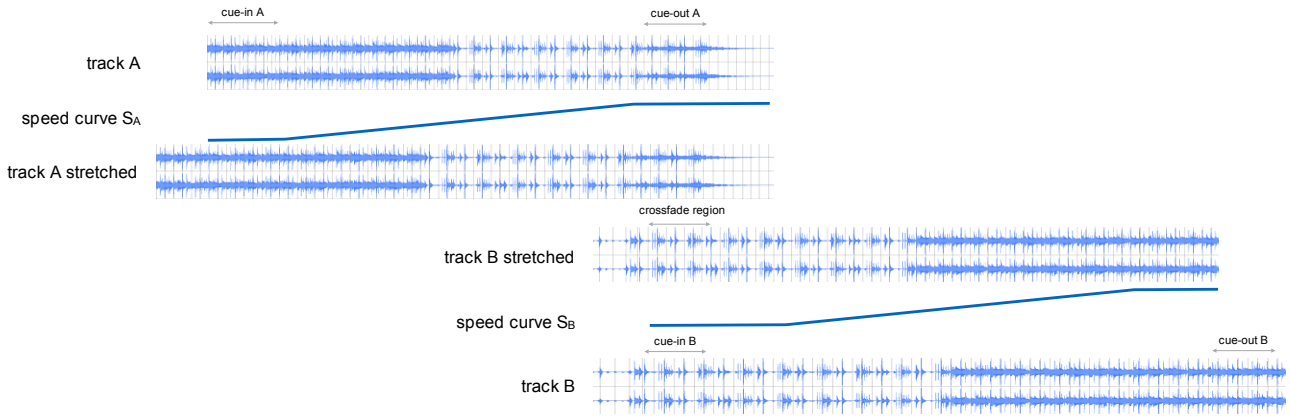
This type of duration compensation was prototyped and demoed in a Max/MSP patch based on the SuperVP time stretching modules by Ircam:




*Screenshot of duration compensation prototype allowing to test different mix speed conditions.*

## 2.2 Beat Alignment Algorithm

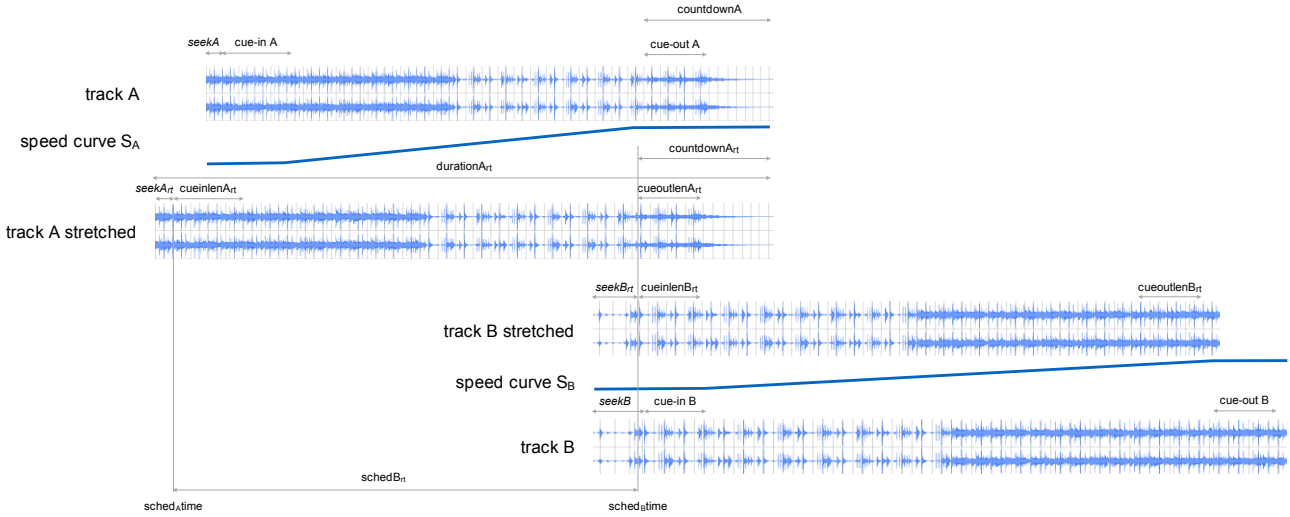
The key idea for beat-synchronous mixing is, first, to apply time-stretching to align the tempo of the cross-fade region of the two tracks to be mixed. This is facilitated by the speed curve model described in the previous section, stipulating a constant speed curve during cross-fade. Second, we determine a synchronisation point within the cue-region to align the beats (see figure).




*[fig:beatalign] Beat alignment and cue-regions for cross-fade between tracks.*

Note that three time scales have to be managed: The source track A's and B's original times, in which the respective cue-regions and beat markers are given, and the actual playing time (*real time*), given by the speed curves applied to both tracks to achieve beat synchronisation and duration compensation.

The beat alignment algorithm was prototyped in JavaScript within the prototyping environment Max/MSP for easy transfer to the ISP implementation in C++. We provide here the equations necessary to calculate the schedule interval *schedB_rt* to start playing the next track B, and the start time in track B *seekB*, given the tracks' bpm, duration, beat markers, and cue-regions, and the current track A's seek time *seekA* and cross-fade region bpm. See the following diagram for an explanation of the different time measures calculated by the algorithm. $\int S_A$ is the function mapping track A's time to real time, depending on the speed curve $S_A$.




*[fig:syncparams] Beat alignment algorithm parameters and measures.*

The formulas needed to calculate the beat-aligned scheduling values are given by the following equations:

$track_X$ with $X = A|B$ contains the metadata of one track, where A is the current track, B is the next track to be mixed with A:

| | |
|---|---|
| $track_X beatmarkers$ | position of beats in track (ms) |
| $track_X bpm$ | speed of track |
| $track_X duration$ | original duration (ms) |
| $track_X cuein$ | cue-in region as list of (starttime, length) in ms |
| $track_X cueout$ | cue-out region as list of (starttime, length) in ms |

Function $fillbeats(segment, beats, bpm)$ returns the list of beats in the given segment, completed according to bpm and position of existing beats.

Function $getsynctimes(cueseg, cuebeats)$ calculates the synchronisation points in the two respective tracks: first (filled) beat or cue start:

$$getsynctimes(cueseg, cuebeats) = \begin{cases} (cueseg_{A0}, cueseg_{B0}) & \text{if } cuebeats_A length = 0 \text{ or } cuebeats_B length = 0 \text{ (if one has no beats: easy, no beat sync necessary, sync on cuein/out start)} \\ (cuebeats_{A0}, cuebeats_{B0}) & \text{otherwise, both segments have beats: fill cue segments with continuing beats according to bpm} \end{cases}$$

with

$$\begin{aligned} cuebeats_A &= fillbeats(cueseg_A, cuebeats_A, track_A bpm) \\ cuebeats_B &= fillbeats(cueseg_B, cuebeats_B, track_B bpm) \end{aligned}$$

Now, given the current position in track A, we can calculate the play data for mixing in track B: the real time schedule interval relative to when track A was started (at $seekA$) $schedB_{rt}$ and the seek time $seekB$, based on the following values:

$$\begin{aligned} curbpm &= \text{current bpm (constant)} \\ seekA &= \text{previous seek time in track A} \\ durA_{rt} &= track_A duration \end{aligned}$$

Note that the length compensation guarantees that track A's duration is maintained despite time-stretching for beat sync mixing. We first calculate the speed factors (track time to real time) and *cueoutA*, *cueinB* (start, length) in track time and real time:

$$\begin{aligned} speedA &= curbpm/track_A bpm \\ speedB &= curbpm/track_B bpm \\ cueoutA &= track_A cueout \\ cueinB &= track_B cuein \\ cueoutlenA_{rt} &= cueoutA_1/speedA \\ cueinlenB_{rt} &= cueinB_1/speedB \end{aligned}$$

final xfade length in real time: minimum of both

$$cuelen_{rt} = min(cueoutlenA_{rt}, cueinlenB_{rt})$$

The final cue segments in track time and cue beats in final xfade period (time relative to tracks) are:

$$\begin{aligned} cueseg = ( & (cueoutA_0, cuelen_{rt} * speedA), \\ & (cueinB0, cuelen_{rt} * speedB)) \\ cuebeats = ( & get\_markers\_within(cueseg_A, track_A beatmarkers), \\ & get\_markers\_within(cueseg_B, track_B beatmarkers)) \end{aligned}$$

We can now align the beats by calculating sync times in respective tracks:

Calculate real time of sync time:

$$synctime = getsynctimes(cueseg, cuebeats)$$

Duration between sync point to end of track A:

$$countdownA = track_A duration - synctime_A$$

…in real time[2]

2 Length compensation guarantees that track duration is maintained despite time-

$$countdownA_{rt} = countdownA/speedA$$

duration of *cueseg* before sync point:

$$precountB = synctime_B - cueseg_{B0}$$

…in real time

$$precountB_{rt} = precountB/speedB$$

last seek time within A in real time

$$seekA_{rt} = seekA/speedA$$

We can now calculate the schedule interval to start playing B at $seekB$ relative to last schedule time (starting to play A at $seekA$) and the seek time in B ($<= synctime_B$) to be started after $schedB_{rt}$.

$$\begin{array}{rl} schedB_{rt} & = durA_{rt} - seekA_{rt} - countdownA_{rt} - precountB_{rt} \\ seekB & = cueseg_{B0} \end{array}$$

An open development question still to be resolved by human-listener evaluation of the algorithm on a more extensive set of tracks is:

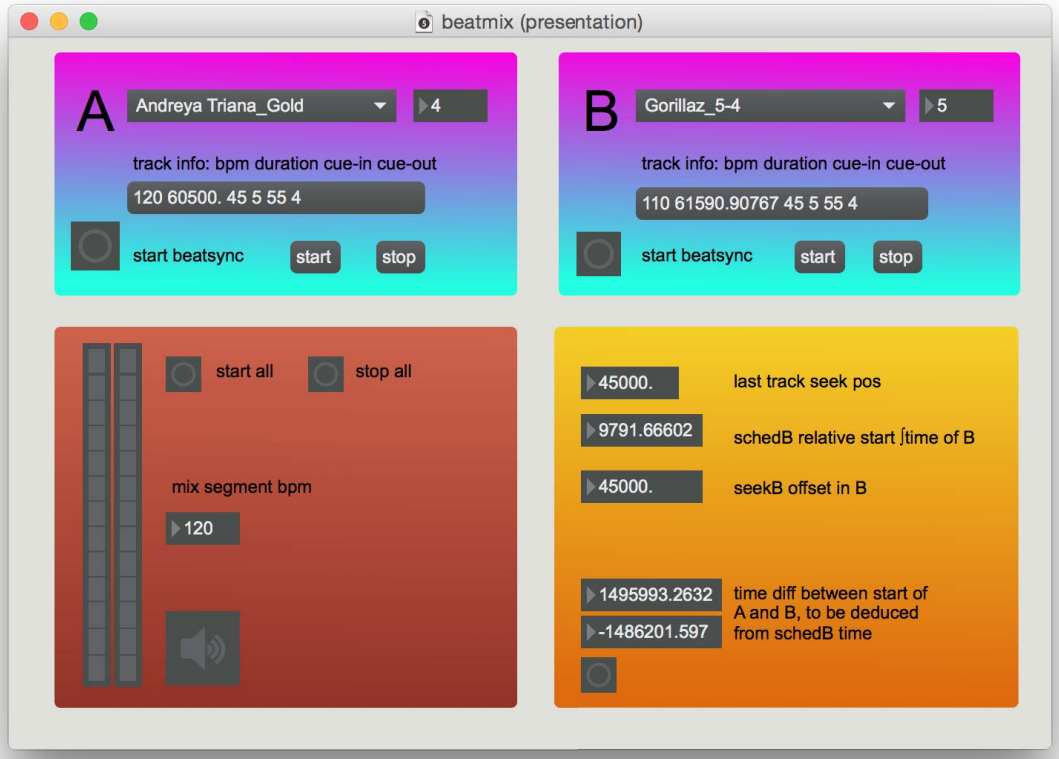
- What to do when one cue-region is a cut (a very short fade-in or fade-out)? For example: when the cue-in is of duration 0, should cue-out fade out normally, fade out quickly, or be cut, too?

## 2.3 Beat-Synchronous Mixing Prototype Software

The prototype for a beat-synchronous mixing patch in Max/MSP is shown below. It allows to choose two tracks from the example set and mix them beat-synchronously using the algorithm described above. It applies high-quality time-stretching using the SuperVP engine by IRCAM, and linear cross-fades.

stretching for beat sync mixing.




*[fig:beatmix] Screenshot of beat-synchronous mixing prototype.*

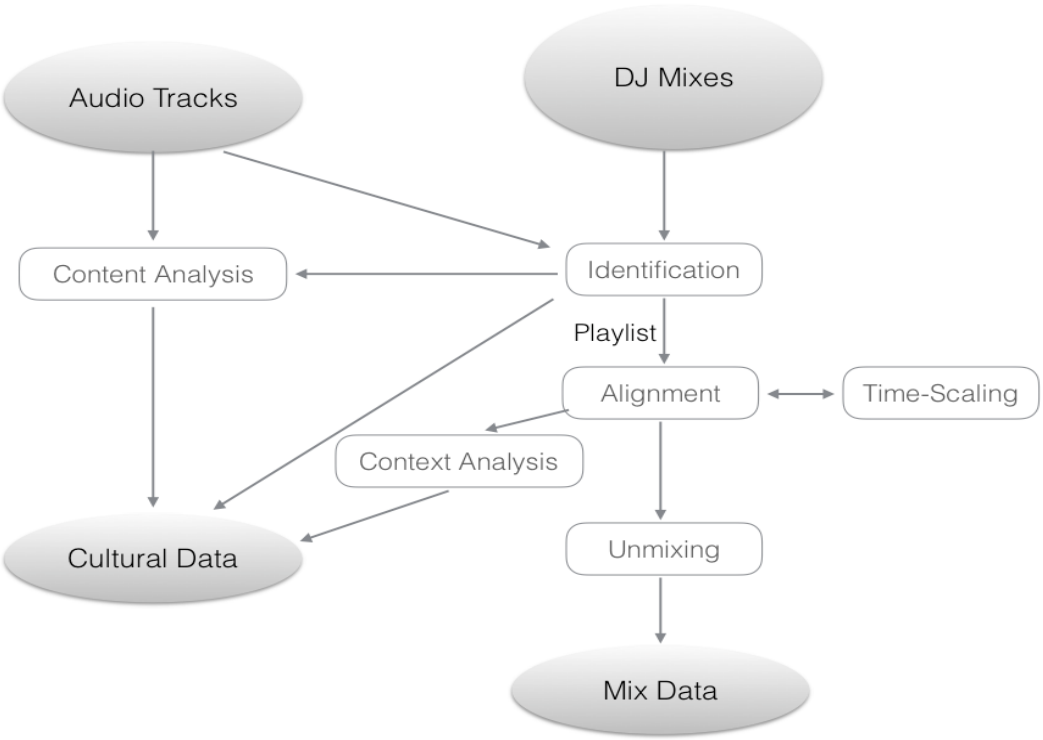
# 3 Unmixing: Extraction of Ground Truth Data from DJ Mixes

A further direction of prospective research motivated by the work in the present WP was to generate a systematic approach for collecting examples of annotated mixes and tracks. In general, there is a scarcity of available ground truth databases that are able to deliver stable hypotheses about usual cross-fade times and styles, and typical DJ practices in general.

A proposal how to tackle this problem technically was presented at ISMIR 2017[3] and ISMIR 2018 and is summarised in the following.

## 3.1 Unmixing Introduction

This work offers one missing brick in a larger research agenda aiming at systematically analysing and understanding DJ practices—an important part of popular music culture. The possible benefits from such efforts are many, for instance musicological research in popular music, cultural studies on DJ practice and reception, development of new music technologies and software products for computer support of DJing, automation of DJ mixing for entertainment or commercial purposes. So far, DJ techniques are not very well researched, not least due to the lack of annotated databases of DJ mixes.




*[fig:schema] Overview of the larger context of information retrieval from DJ practices.*

3 Diemo Schwarz, Dominique Fourer. Towards Extraction of Ground Truth Data from DJ Mixes. International Symposium on Music Information Retrieval (ISMIR), Oct 2017, Suzhou, China. hal-01671768

In order to be able to annotate recorded mixes automatically, several components are needed, see the overview figure [fig:schema].

**Identification**
of the contained tracks (e.g. by acoustic fingerprinting) to obtain the playlist,

**Alignment**
to determine where in the mix each track starts and stops

**Time-scaling**
to determine what tempo changes were applied to achieve beat-synchronicity,

**Unmixing**
to estimate the cue-regions where the cross-fades between tracks happen, the curves for volume, bass and treble, and the parameters of other effects (compression, echo, etc.)

**Content and metadata analysis**
to derive the genre and social tags attached to the music to inform about the choices a DJ makes when creating a mix.

Most of these necessary future components have been addressed by recent MIR research, except the alignment part for which we will provide here a first attempt that draws on multi-scale correlation and dynamic time warping (DTW) techniques.

With some refinements, a massive amount of training data from the vast number of collections of existing DJ mixes could be made amenable to musicological research on DJ practices, cultural studies, and for development of automatic mixing methods.

As a working definition, it is possible to roughly distinguish 3 levels of mixing:

- Level 1, let's call this *broadcast mixing*, is a simple volume crossfade without paying attention to, or changing content (as it is performed by consumer audio players such as iTunes, or in a broadcast context).
- Level 2, *lounge mixing*, is beat-synchronous mixing with adaptation of the tempo of the tracks and possibly additional EQ fades, while playing the tracks mostly unchanged.
- Level 3, *performative mixing*, is using the DJ deck as a performance instrument by creative use of effects, loops, and mashups with other tracks.

Our work applies to *broadcast* or *lounge mixing* with simple crossfades, while *performative DJ mixing* tends to blur the identifiability of the source tracks too much.

## 3.2 Related Work

First of all, there is much more existing research in the field of *studio mixing*, where a stereo track is to be produced from individual multi-track recordings and software

instruments by means of a mixing desk or DAW[4,5,6,7]. This research field has produced ground truth databases[8] and has some overlap with DJ mixing, when we see the latter as mixing just two source tracks, but the studied parameters and influencing factors differ too much from what is needed for DJ mixing.

Furthermore, there are some existing research works on tools to help DJs produce mixes[9,10,11,12,13], but much less regarding information retrieval from recorded mixes.

The work of Sonnleitner, Arzt and Widmer that has been opening up research on information retrieval from DJ mixes[14] tackles the identification of the tracks within the mix by fingerprinting. The authors also produce an extensive database of ground truth annotations of playlists with approximate start and stop times of tracks on a large number of Creative-Commons licensed mixes made from open-licensed dance tracks published on the *Mixotic* net label.[15] This dataset[16] provides 10 dance music mixes with a total duration of 11 hours and 23 minutes, the 118 source tracks, and the playlists with hand-annotated

4 Enrique Perez-Gonzalez and Joshua Reiss. Automatic gain and fader control for live mixing. In Applications of Signal Processing to Audio and Acoustics, 2009. WASPAA'09. IEEE Workshop on, pages 1–4. IEEE, 2009.

5 Jacob A Maddams, Saoirse Finn, and Joshua D Reiss. An autonomous method for multi-track dynamic range compression. In Proceedings of the 15th International Conference on Digital Audio Effects (DAFx-12), 2012.

6 Stuart Mansbridge, Saorise Finn, and Joshua D Reiss. An autonomous system for multitrack stereo pan positioning. In Audio Engineering Society Convention 133. Audio Engineering Society, 2012.

7 Brett Brecht De Man, R; King, and J. D. Reiss. An analysis and evaluation of audio features for multitrack music mixtures. In ISMIR, 2014.

8 Brecht De Man, Mariano Mora-Mcginity, György Fazekas, and Joshua D Reiss. The open multitrack testbed. In Audio Engineering Society Convention 137. Audio Engineering Society, 2014.

9 Hiromi Ishizaki, Keiichiro Hoashi, and Yasuhiro Takishima. Full-automatic DJ mixing system with optimal tempo adjustment based on measurement function of user discomfort. In ISMIR, pages 135–140, 2009.

10 Dave Cliff. Hang the DJ: Automatic sequencing and seamless mixing of dance-music tracks. HP Laboratories Technical Report HPL, 104, 2000.

11 Tsuyoshi Fujio and Hisao Shiizuka. A system of mixing songs for automatic dj performance using genetic programming. In 6th Asian Design International Conference, 2003.

12 Felipe X Aspillaga, Jonathan Cobb, and Ching-Hua Chuan. Mixme: A recommendation system for djs. In Late-break Session of the 12th International Society for Music Information Retrieval Conference, 2011.

13 Pablo Molina, Martin Haro, and Sergi Jorda. Beat-jockey: A new tool for enhancing DJ skills. In NIME, pages 288–291. 2011.

14 Reinhard Sonnleitner, Andreas Arzt, and Gerhard Widmer. Landmark-based audio fingerprinting for DJ mix monitoring. In ISMIR, New York, NY, 2016.

15 http://www.mixotic.net

16 http://www.cp.jku.at/datasets/fingerprinting

time points relevant for fingerprinting, namely the moment from which only the next track in the playlist is present in the mix. Unfortunately, this dataset does not give information about the start point of the track in the mix, and is not accurate enough for our aims of DJ mix analysis, let alone reverse engineering.

Barchiesi and Reiss[17] first used the term *mix reverse engineering* (in the context of multi-track studio mixing) for their method to invert linear processing (gains and delays, including short FIR filters typical for EQ) and some dynamic processing parameters (compression).

Ramona and Richard[18] tackle the unmixing problem for radio broadcast mixes, i.e. retrieving the fader positions of the mixing desk for several known input signals (music tracks, jingles, reports), and one unknown source (the host and guests' microphones in the broadcast studio). They model the fader curves as a sigmoid function and assume no time-varying filters, and no speed change of the sources (which is correct in the context of radio broadcast practice).

These two latter references both assume having sample-aligned source signals at their disposal, with no time-scaling applied, unlike our use-case, where each source track only covers part of the mix, can appear only partially, and can be time-scaled for beat-matched mixing.

There is at present only rare work on the inversion of other processing applied to the signal[19], notably compression[20].

## 3.3 DJ Mix Reverse Engineering

The starting point for our method is the result of the previous stage of identification and retrieval on existing DJ mixes (see [fig:schema]), or specially contrived databases for the study of DJ practices: We assume a recorded DJ mix, a playlist (the list of tracks played in the correct order), and the audio files of the original tracks.

Our method proceeds in five steps, from a rough alignment of the concatenated tracks with the mix by dynamic time warping (DTW), that is then refined in order to approach sample precision, then verified by subtracting the track out of the mix, and is then finally extended by the estimation of gain curves and cue-regions.

### Step 1: Rough Alignment

The rough alignment uses the MFCC data of the mix and the concatenated MFCCs of the tracks as input. We use the MIRToolbox[21] MFCC with 13 coefficients, a window size of 0.05

17 Daniele Barchiesi and Joshua Reiss. Reverse engineering of a mix. Journal of the Audio Engineering Society, 58(7/8):563–576, 2010

18 Mathieu Ramona and Gael Richard. A simple and efficient fader estimator for broadcast radio unmixing. In Proc. DAFX '11, pages 265–268, September 2011.

19 Stanislaw Gorlow and Sylvain Marchand. Reverse engineering stereo music recordings pursuing an informed two-stage approach. In 2013 International Conference on Digital Audio Effects (DAFx-13), pages 1–8, 2013.

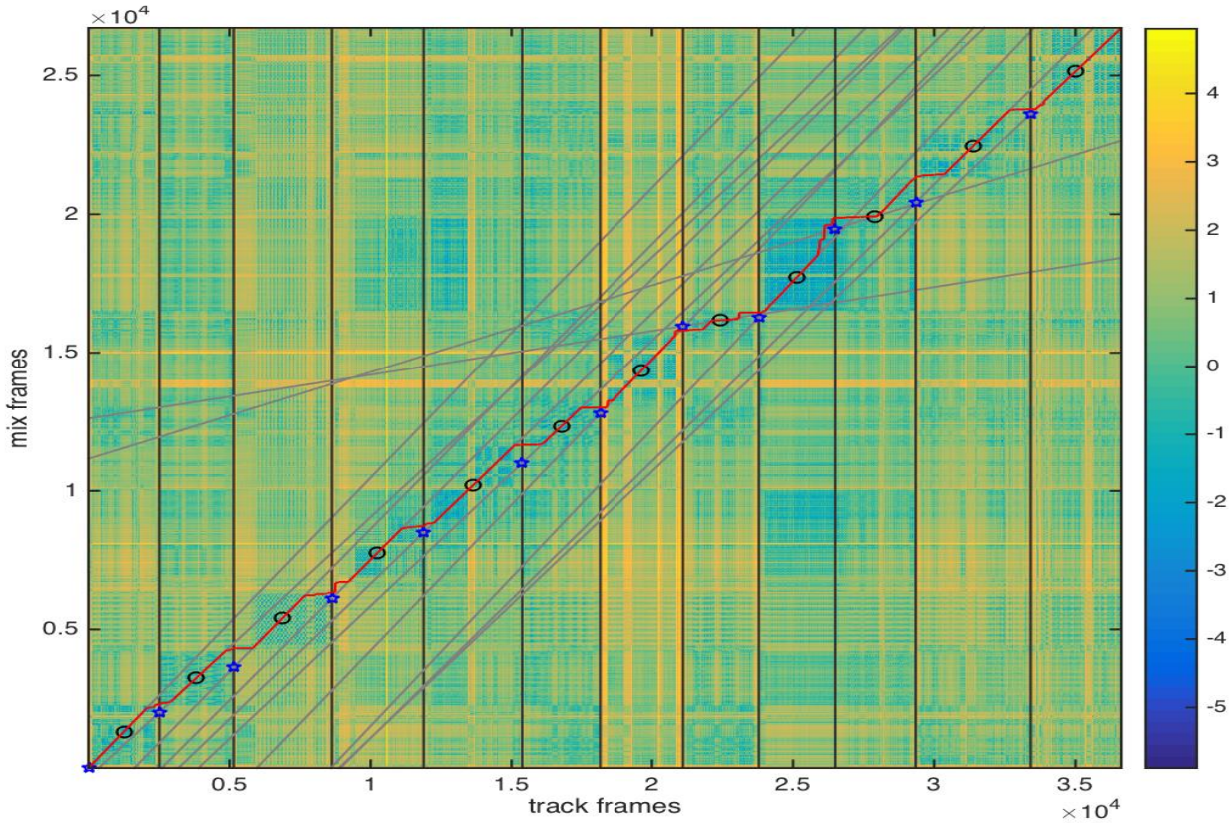
20 Stanislaw Gorlow and Joshua D Reiss. Model-based inversion of dynamic range compression. IEEE Transactions on Audio, Speech, and Language Processing, 21(7):1434–1444, 2013

21 Olivier Lartillot, Petri Toiviainen, and Tuomas Eerola. A matlab toolbox for music

s and a small hop size of 0.0125 s. We chose the Mel frequency band representation because, compared to an FFT representation, it is more robust against pitch changes due to resampling of the source tracks.

Since the tracks are largely unchanged in terms of the level 2 mixes we are interested in, DTW can latch on to large valleys of low distance, although the fade regions are dissimilar to either track, and occur separately in the concatenated track MFCC stream (see [fig:dtw]). To ease catching up with the shorter time of the mix, we provide a neighbourhood that allows the path $p$ to perform larger vertical and horizontal jumps as follows, given the initial condition $p(1,1) = d(1,1)$ on the local distance matrix $d$:

$$p(m,n) = \min \left\{ \begin{array}{ccc} p(m-1,n-1) & + & d(m,n) \\ p(m-1,n) & + & d(m,n) \\ p(m-2,n) & + & d(m,n) \\ p(m,n-1) & + & d(m,n) \\ p(m,n-2) & + & d(m,n) \end{array} \right\}$$




*[fig:dtw] DTW distance matrix, alignment path (red), track boundaries and found slope lines on a complete dance DJ mix.*

The DTW alignment path not only gives us the relative positioning of the tracks in the mix, but also their possible speed up or slow down to achieve beat-synchronous mixing or smoother evolution of the tempo of the mix. To retrieve the tempo change, we assume that the change is constant for each track, and analyse the mean slope of the alignment path in a window of half the track length, centred around the middle of the track. This also gives

information retrieval. In Data analysis, machine learning and applications, pages 261–268. Springer, 2008.

us an estimate of the start of the track in the mix, by calculating the intercept of the slope for the start frame of the track (see [fig:dtw]).

Note that that start position expresses the offset of the start of the full source track with respect to the mix, and not the point from where on the track is present in the mix. Since the source tracks are mixed with non-zero volume only between the cue-in and cue-out regions, the track start point can be negative. (See section *The UnmixDB Dataset* for an explanation how the cue-regions of our dataset are chosen.)

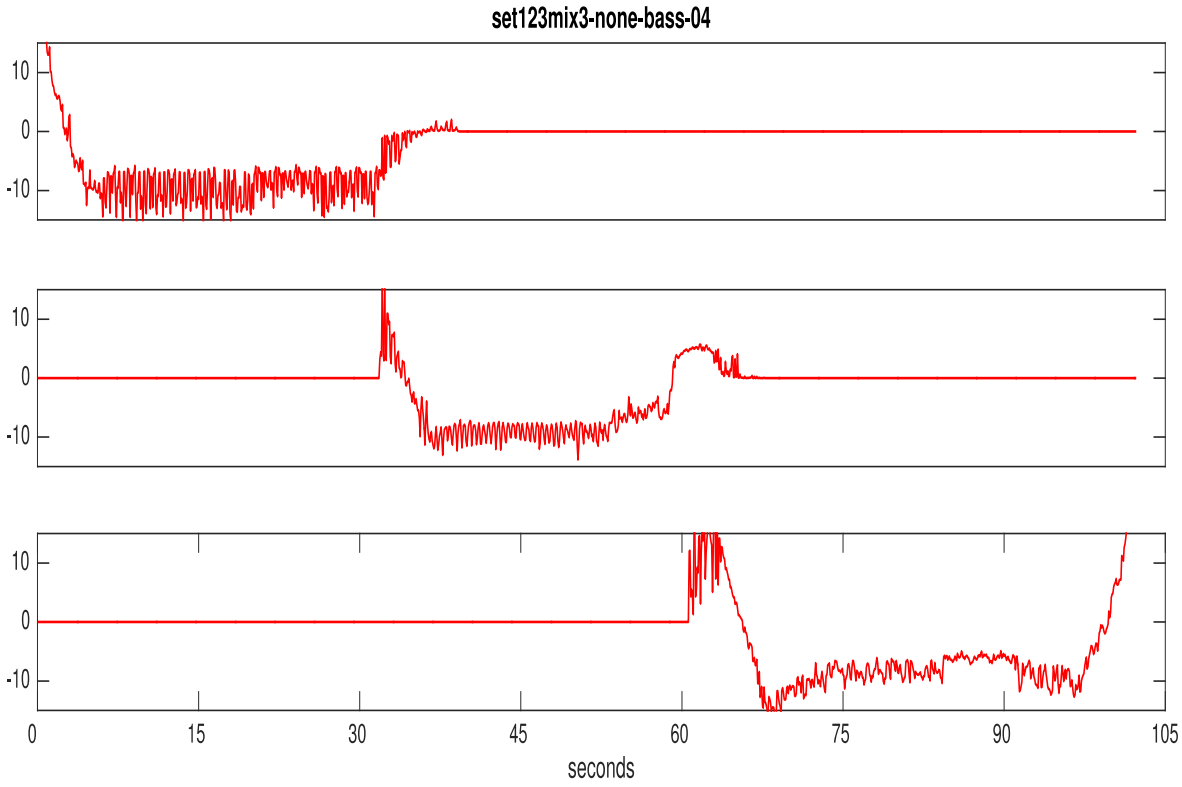
### Step 2: Sample Alignment

Given the rough alignment and tempo estimation by DTW, we then search for the best sample alignment of the source tracks time-scaled according to the estimated tempo. We shift a window of the size of an MFCC frame, taken from the middle of the time-scaled track, around its predicted rough frame position in the mix, trying displacements up to 2 hop sizes in either direction. The minimum sum of square distances then determines the sample alignment.

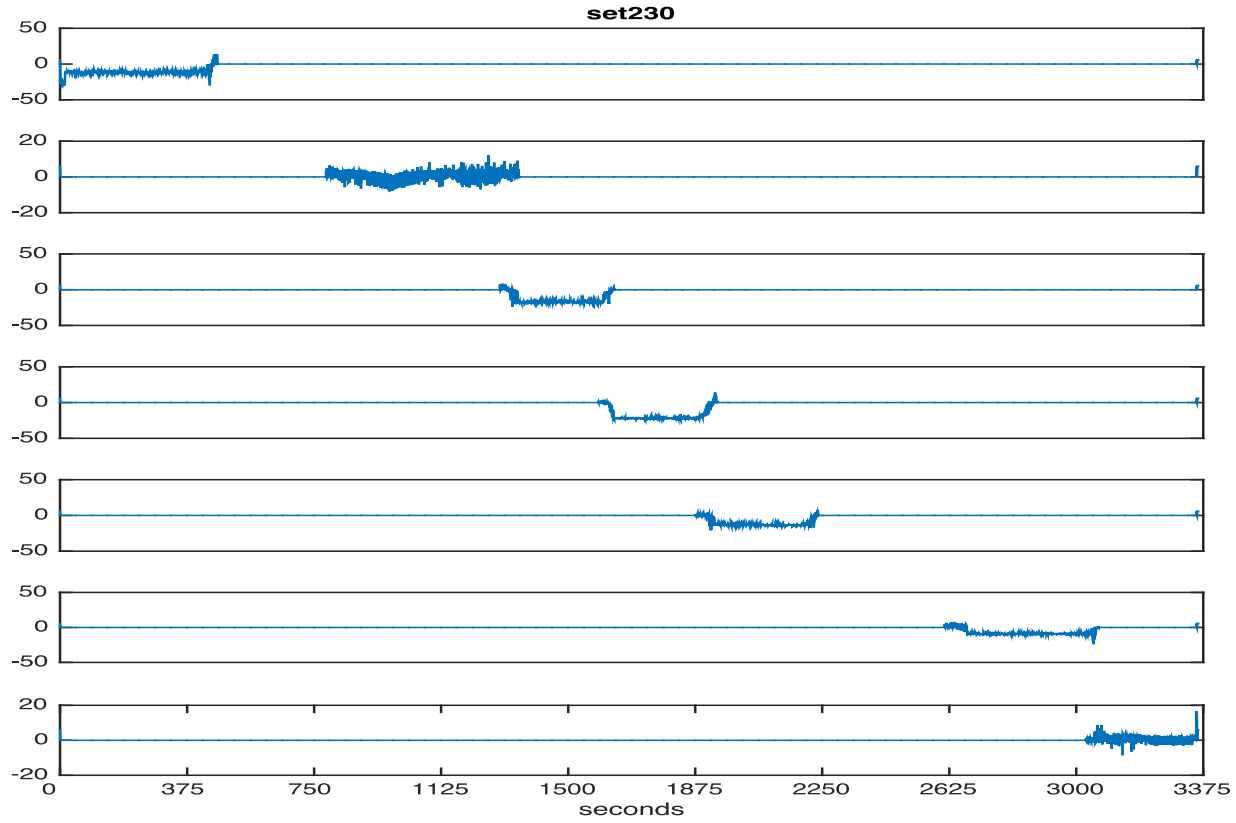
### Step 3: Track Removal

The accuracy of the resulting sample alignment can be verified by attempting to remove the aligned and time-scaled track from the mix: For this we need to subtract its signal from the mix and observe the resulting drop in energy. Note that this verification of sample alignment can be done even when the ground truth is not given or inexact.

Figure [fig:remove 1] shows this on one mix from our dataset: we can observe that for all tracks the mix energy shows a drop of around 10 dB. Because of the missing sub-sample alignment, mainly the low-frequency material is suppressed. Another example is figure [fig:remove 2] on a mix from Sonnleitner et. al., where we can immediately see which tracks were not correctly aligned (the second and last ones).




*[fig:remove 1] Suppression of tracks from a mix in dB.*




*[fig:remove 2] Suppression of tracks from a complete mix in dB.*

## Step 4: Volume Curve Estimation

We propose here a novel method which operates in the time-frequency plane to estimate the volume curve that is applied to each track in order to obtain the mix. Given a mix signal denoted $x(n)$ and the constituent sample-aligned and time-scaled tracks $s_i(n)$, we aim at estimating the mixing function $a_i(n)$ such that we have:

$$x(n) = \sum_{i} a_i(n) s_i(n) + b(n) \quad \forall n \in \mathbf{Z}$$

where $b(n)$ corresponds to an additional noise signal.

Considering a "correctly" aligned track $s_i$, we estimate its corresponding volume curve $\hat{a}_i$ through the following steps:

1. compute the short-time Fourier transforms of $x$ and $s_i$ denoted $S_i(n,m)$ and $X(n,m)$ ($n$ and $m$ being respectively the time and frequency indices)
2. estimate the volume curve at each instant $n$ by computing the median of the mix/track ratio computed at all the frequencies $m'$ where $S_i(n,m')$ contains energy, such as:

$$\hat{a}_i(n) = \begin{cases} \text{median}\left(\frac{|X(n,m')|}{|S_i(n,m')|}\right)_{\forall m'} & \text{if} |S_i(n,m')| > 0 \\ 0 & \text{otherwise} \end{cases}$$

3. post-process $\hat{a}_i(n)$ to obtain a smooth curve by removing outliers using a median filter.

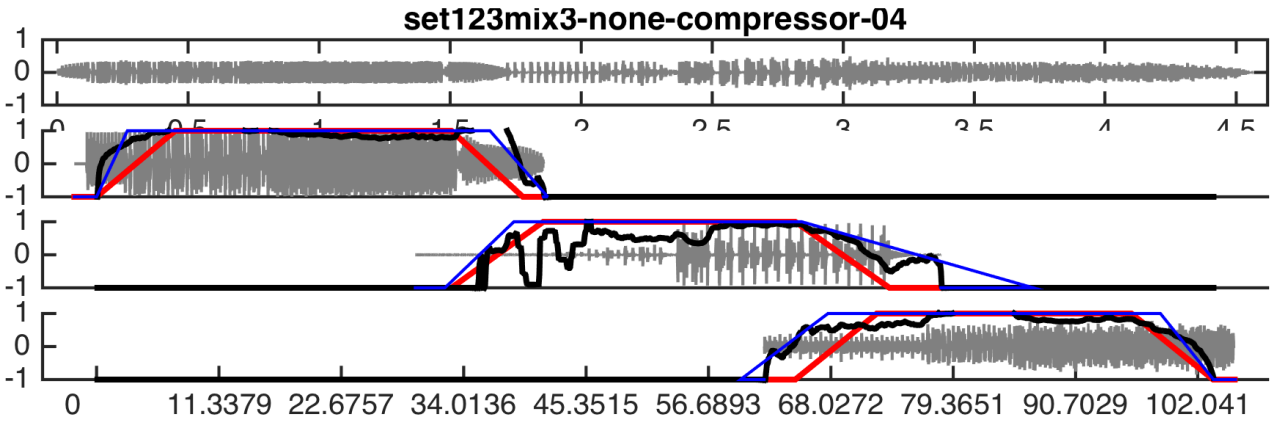
The resulting volume curve can then be used to estimate the cue points (the time instants when a fading effect begins or stops) at the next step.

### Step 5: Cue Point Estimation

In order to estimate the DJ cue points, we apply a linear regression of $\hat{a}_i$ at the time instants located at the beginning and at the end of the resulting volume curve (when $\hat{a}_i(n) < \Gamma$, $\Gamma$ being a threshold defined arbitrarily as $\Gamma = 0.7\max(\hat{a})$). Assuming that a linear fading effect was applied, the cue points can easily be deduced from the two affine equations resulting from the linear regression. The four estimated cue points correspond respectively to:

1. the time instant when the fade-in curve is equal to 0
2. the time instant when the fade-in curve is equal to $\max(\hat{a}_i)$
3. the time instant when the fade-out curve is equal to $\max(\hat{a}_i)$
4. the time instant when the fade-out curve is equal to 0.

In order to illustrate the efficiency of the entire method (steps 4 and 5), we present in Fig. [fig:fading_cuepoint] the results obtained on a real-world DJ-mix extracted from our proposed dataset.




*[fig:fading_cuepoint] Estimated volume curve (black), linear fades (blue), over ground truth fades (red)*

## 3.4 The *UnmixDB* Dataset

In order to evaluate the DJ mix analysis and reverse engineering methods described above, we created a dataset of excerpts of open licensed dance tracks and automatically generated mixes based on these. This dataset was uploaded to the OpenAIRE-indexed open data repository Zenodo[22], and was presented at the ISMIR 2018 late breaking session[23].

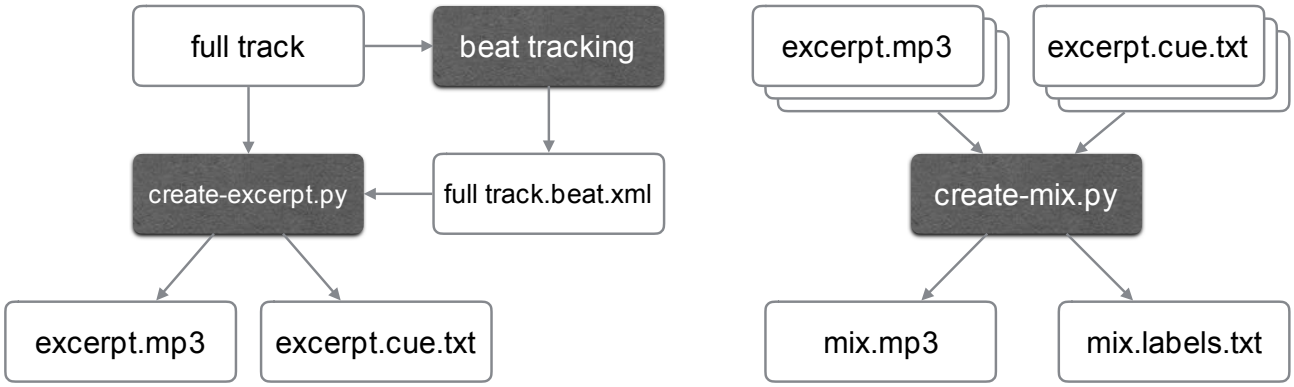
Possible uses of the dataset are the evaluation of track identification methods when monitoring DJ mixes, or the precise annotation or even reverse engineering of DJ-mixes when constituent tracks are available. In the latter project, we perform alignment to determine the exact offset of each track in the mix, and then estimate cue points and volume fade curves, in order to learn about the decisions a DJ makes when creating a mix.

The *UnmixDB* dataset is based on the curatorial work of Sonnleitner et. al., for identification of the tracks within human-made DJ mixes by fingerprinting. They collected Creative-Commons licensed source tracks of 10 free dance music mixes from the *Mixotic*

[22] http://zenodo.org/record/1422385
[23] D. Schwarz and D. Fourer, "Unmixdb: A Dataset for DJ-Mix Information Retrieval," International Symposium on Music Information Retrieval (ISMIR) late breaking session, Paris, France, Sep. 2018.

netlabel.[24] Their dataset[25] provides the mixes, the full tracks, and the ground-truth playlists with hand-annotated time points from which only the next track in the playlist is present in the mix. Unfortunately, this does not give information about the start point of the track in the mix, and is not accurate enough for our aims of DJ mix analysis, let alone reverse engineering.

We used their collected full tracks to produce our track excerpts, but regenerated beat-synchronous and thus "ecologically valid" artificial mixes with perfectly accurate ground truth, see [fig:unmixdb-creation].




*[fig:unmixdb-creation] Data flow and file types of the UnmixDB dataset.*

We used track excerpts because of the runtime and memory requirements, especially for methods such as DTW, which is of quadratic memory complexity. We could also not have scaled the dataset up to the many playlists and variants when using full tracks.

Each track excerpt contains about 20 s of the beginning and 20s of the end of the full source track (not included in the dataset, but available from Sonnleitner et. al.[26]). However, the exact choice is made taking into account the metric structure of the track. The cue-in region, where the fade-in will happen, is placed on the second beat marker starting a new measure (as analysed by the beat tracker IrcamBeat), and lasts for 4 measures. The cue-out region ends with the 2nd to last measure marker. We assure at least 20s for the beginning and end parts by extending them accordingly. The cut points where they are spliced together is again placed on the start of a measure, such that no artefacts due to beat discontinuity are introduced.

Each mix is based on a playlist that mixes 3 track excerpts beat-synchronously, in such a way, that the middle track is embedded in the realistic context of beat-aligned linear cross fading to the surrounding tracks. The first track's BPM is used as the tempo seed onto which the other tracks are adapted.

Each playlist of 3 tracks was mixed 12 times with every combination of 4 variants of effects and 3 variants of time scaling using the treatments of the *sox* open source command-line program.[27] The 4 effects were:

| | |
|---|---|
| **none** | no effect |
| **bass** | +6 dB bass boost using a low-shelving biquad filter below 100 Hz |

[24] http://www.mixotic.net
[25] http://www.cp.jku.at/datasets/fingerprinting
[26] Reinhard Sonnleitner, Andreas Arzt, and Gerhard Widmer. Landmark-based Audio Fingerprinting for DJ Mix Monitoring. In *Proc. ISMIR*, New York, NY, 2016.
[27] http://sox.sourceforge.net

| | |
|---|---|
| **compressor** | heavy dynamics compression (ratio of 3:1 above -60 dB, -5 dB makeup gain) |
| **distortion** | heavy saturation with +20 dB gain |

These effects were chosen to cover treatments likely to be applied to a DJ set (EQ, compression), and also to introduce non-linear treatments (distortion) to test the limits of MIR methods.

The 3 timescale variants were:

| | |
|---|---|
| **none** | no time scaling, ie. the tracks are only aligned on the first beat in the cue region and then drift apart |
| **resample** | linked time and pitch scaling by resampling (sox *speed* effect) |
| **stretch** | time stretching while keeping the pitch (sox *tempo* effect using WSOLA) |

These 3 variants allow to test simple alignment methods not taking into account time scaling, and allow to evaluate the influence of different algorithms and implementations of time scaling.

The dataset is organised in 6 individually downloadable sets of tracks and mixes, between 500 MB and 1 GB in size, for a total of 4 GB. Table [tab:dbstats] provides more details about the initial content of one of the sets. In the near future, the dataset may be extended by more songs, more mixes, and mixes of the full source tracks. We also publish the Python source code[28] to generate the excerpts and mixes, such that other researchers can create test data from other track collections or in other variants.

| | |
|---|---|
| **Average duration of mixes [s]** | 107 |
| **Total duration of tracks [min]** | 1016 |
| **Total duration of mixes [min]** | 2743 |
| **Median tempo of tracks [bpm]** | 128 |
| **Minimum tempo of tracks [bpm]** | 67 |
| **Maximum tempo of tracks[bpm]** | 140 |

*[tab:dbstats] Basic statistics of the Unmixdb dataset.*

## File Formats

The *UnmixDB* dataset contains the ground truth for the source tracks and mixes in `.labels.txt` files with tab-separated columns *starttime, endtime, label*. For each mix, the start, end, and cue points of the constituent tracks are provided, along with their BPM and speed factors. We use the convention that the label starts with a number indicating which of the 3 source tracks the label refers to.

The song excerpts are accompanied by their cue region and tempo information in `.txt` files in table format.

Additionally, we provide the `.beat.xml` files containing the beat tracking results for the full tracks available from Sonnleitner et. al.

[28] http://github.com/Ircam-RnD/unmixdb-creation

## 3.5 Evaluation

We applied the DJ mix reverse engineering method on our *unmixdb* collection of mixes and compared the results to the ground truth annotations. To evaluate the success of our method we defined the following error metrics:

**frame error:**
absolute error in seconds between the frame start time found by the DTW rough alignment (step 1) and the ground truth (virtual) track start time relative to the mix

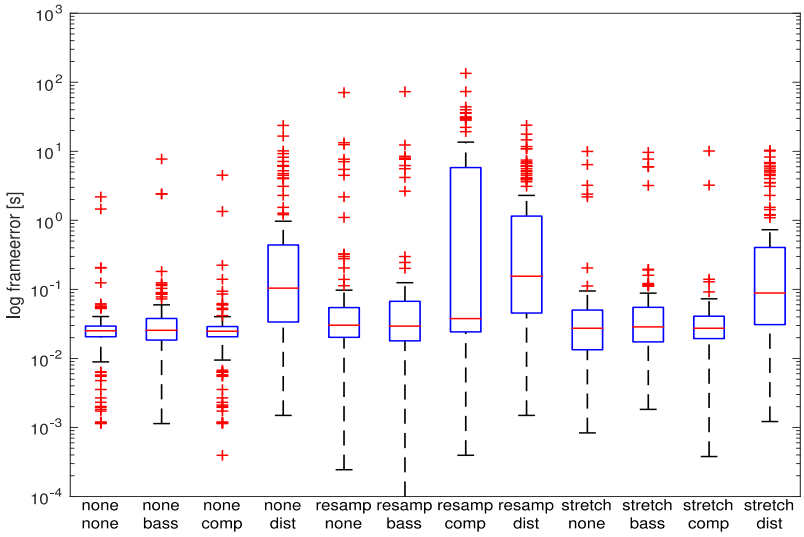
**sample error:**
absolute error in seconds between the track start time found by the sample alignment (step 2) and the ground truth track start time relative to the mix

**speed ratio:**
ratio between the speed estimated by DTW alignment (step 1) and the ground truth speed factor (ideal value is 1)

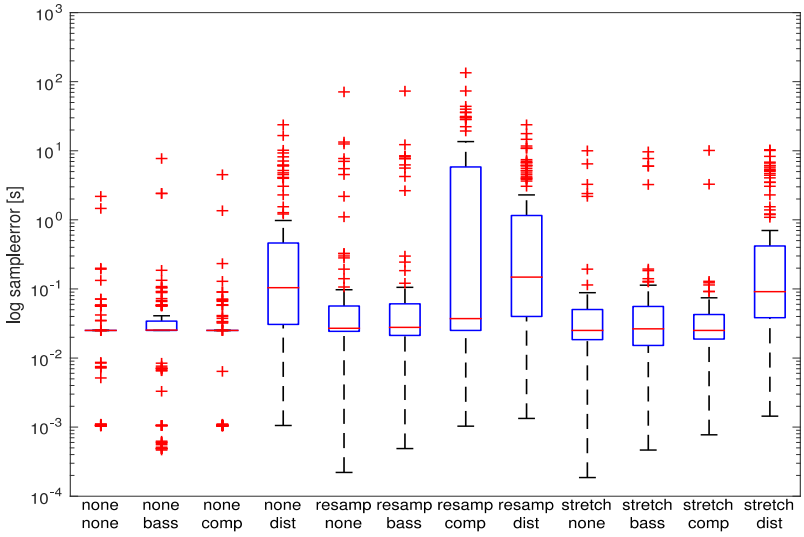
**suppression ratio:**
ratio of time where more than 15 dB of signal energy could be removed by subtracting the aligned track from the mix, relative to the time where the track is fully present in the mix, i.e. between fade-in end and fade-out start (step 3, bigger is better)

**fade error:**
the total difference between the estimated fade curves (steps 4 and 5) and the ground truth fades. This can be seen as the surface between the 2 linear curves over their maximum time extent. The value has been expressed in dB•s, i.e. for one second of maximal difference (one curve full on, the other curve silent), the difference would be 96 dB.

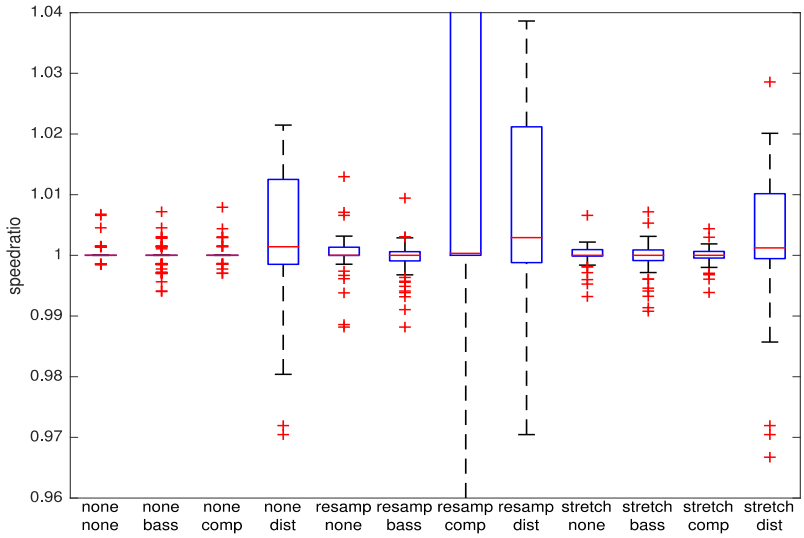
Figures [fig:frameerror]–[fig:fadeerror] show the quartile statistics of these metrics, broken down by the 12 mix variants (all combinations of the 4 mix effects and 3 time-scaling methods). The numeric alignment results given in [tab:sampleerror] show that the ground truth labels can be retrieved with high accuracy: the median error is 25 milliseconds, except for the mixes with distortion applied, where it is around 100 ms. The fade curve volume error in [fig:fadeerror] shows a median of 5 dB•s, which corresponds to an average dB distance of 0.3 dB, considering that the fades typically last for 16 seconds.



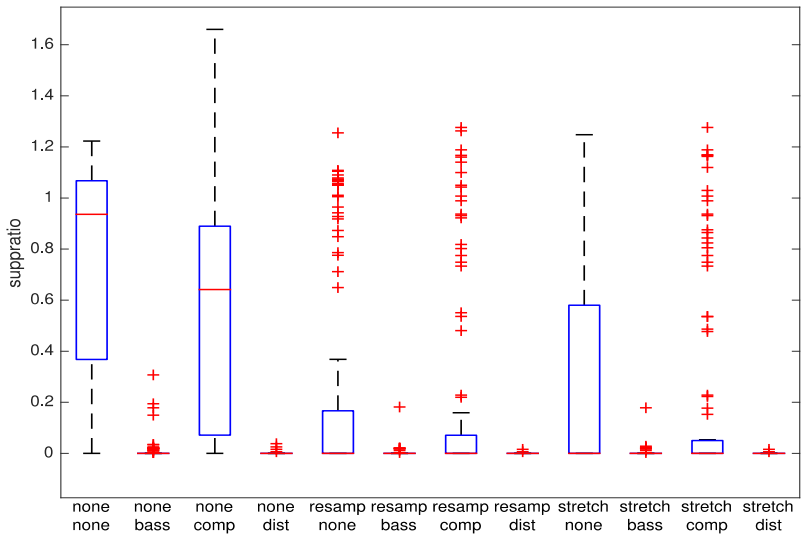

*[fig:frameerror] absolute error in track start time found by DTW*




*[fig:sampleerror] absolute error in track start time found by sample alignment*

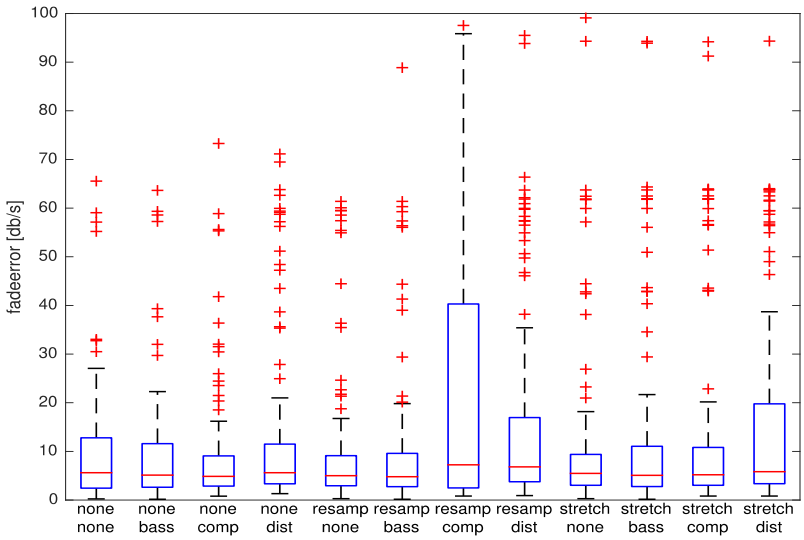



*[fig:speedratio] ratio between estimated and ground truth speed*

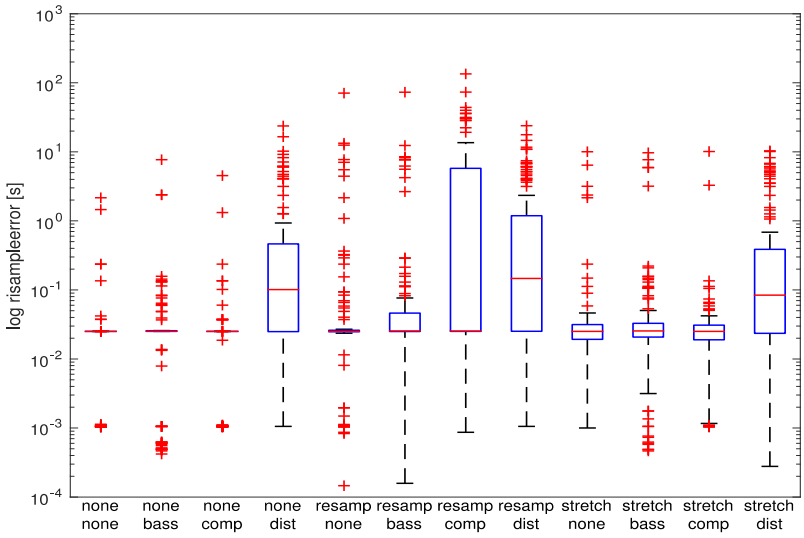



*[fig:suppratio] ratio of removal time (bigger is better)*

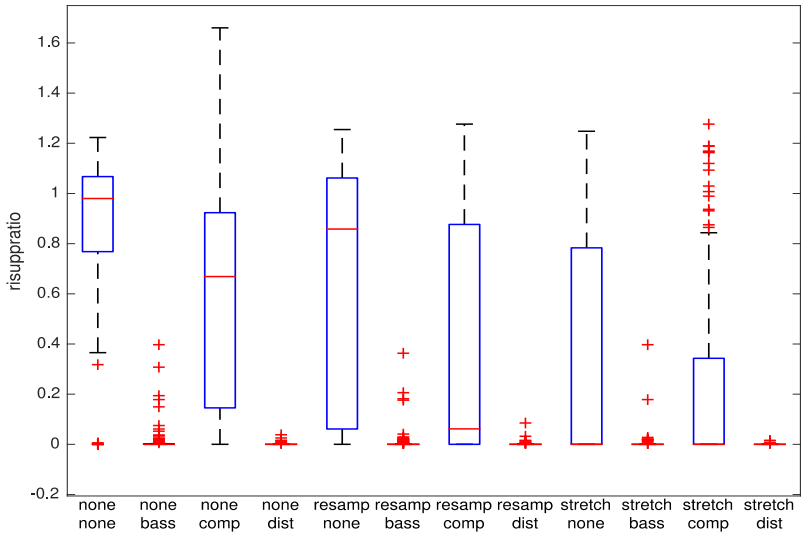
 




*[fig:fadeerror] volume difference of fades*




*[fig:risampleerror] absolute error in track start time found by sample alignment when re-injecting ground truth speed*




*[fig:risuppratio] ratio of removal time when re-injecting ground truth speed (bigger is better)*

| | mean | std | min | median | max |
|---|---|---|---|---|---|
| **none none** | 0.0604 | 0.2469 | 0.0010 | 0.0251 | 2.1876 |
| **none bass** | 0.1431 | 0.7929 | 0.0005 | 0.0254 | 7.7191 |
| **none compressor** | 0.0806 | 0.4424 | 0.0010 | 0.0251 | 4.4995 |
| **none distortion** | 1.3376 | 3.3627 | 0.0011 | 0.1042 | 23.7610 |
| **resample none** | 1.1671 | 7.0025 | 0.0002 | 0.0270 | 71.0080 |
| **resample bass** | 1.3337 | 7.2079 | 0.0005 | 0.0277 | 73.1192 |
| **resample compressor** | 6.8024 | 17.0154 | 0.0010 | 0.0372 | 134.2811 |
| **resample distortion** | 1.8371 | 3.8551 | 0.0013 | 0.1483 | 23.8355 |
| **stretch none** | 0.2502 | 1.1926 | 0.0002 | 0.0251 | 10.0048 |
| **stretch bass** | 0.3300 | 1.4249 | 0.0005 | 0.0264 | 9.6626 |
| **stretch compressor** | 0.1520 | 1.0025 | 0.0008 | 0.0251 | 10.1076 |
| **stretch distortion** | 1.0629 | 2.2129 | 0.0014 | 0.0911 | 10.3353 |
| **all** | 1.2131 | 6.2028 | 0.0002 | 0.0282 | 134.2811 |

*[tab:sampleerror] Statistics of absolute error in track start time found by sample alignment.*

| | mean | std | min | median | max |
|---|---|---|---|---|---|
| **none none** | -0.0025 | 0.0122 | -0.0341 | 0.0000 | 0.0430 |
| **none bass** | -0.0025 | 0.0209 | -0.0549 | 0.0000 | 0.0919 |
| **none compressor** | -0.0022 | 0.0133 | -0.0531 | 0.0000 | 0.0449 |
| **none distortion** | -0.0030 | 0.0359 | -0.0877 | 0.0000 | 0.0931 |
| **resample none** | -0.0063 | 0.0225 | -0.0737 | -0.0011 | 0.0848 |
| **resample bass** | -0.0012 | 0.0272 | -0.0779 | 0.0000 | 0.0919 |
| **resample compressor** | -0.0049 | 0.0343 | -0.0999 | 0.0000 | 0.0970 |
| **resample distortion** | -0.0049 | 0.0346 | -0.0857 | -0.0042 | 0.0931 |
| **stretch none** | -0.0072 | 0.0208 | -0.0985 | -0.0012 | 0.0430 |
| **stretch bass** | -0.0029 | 0.0265 | -0.0768 | 0.0000 | 0.0919 |
| **stretch compressor** | -0.0059 | 0.0180 | -0.0574 | 0.0000 | 0.0385 |
| **stretch distortion** | -0.0068 | 0.0341 | -0.0779 | -0.0043 | 0.0961 |
| **all** | -0.0042 | 0.0262 | -0.0999 | 0.0000 | 0.0970 |

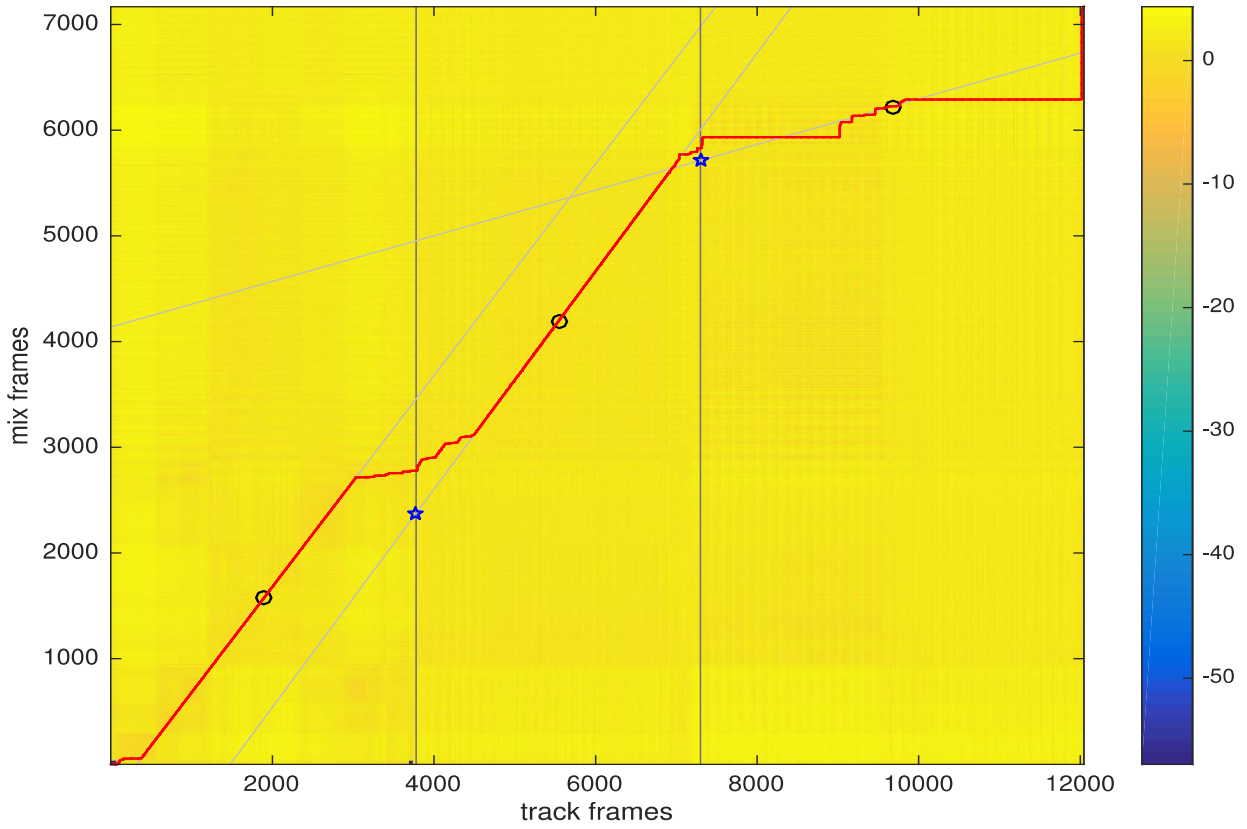
*[sample_to_risample] Improvement of sample-alignment error when reinjecting ground truth speed.*

While developing our method, we noticed the high sensitivity of the sample alignment and subsequent track removal (steps 2 and 3) on the accuracy of the speed estimation. This is due to the resampling of the source track to match the track in the mix prior to track

removal. An estimation error of a tenth of a percent already results in desynchronisation after some time. To estimate this loss in accuracy, we produced a second set of the *sample error* and *suppression ratio* metrics based on a run of steps 2 and 3 with the ground truth speed re-injected into the processing. The rationale is that the speed estimation method could be improved in future work, if the resulting reductions of error metrics are worthwhile. Also note that the tempo estimation is inherently inaccurate due to it being based on DTW's discretisation into MFCC frames. In mixes with full tracks, the slope can be estimated more accurately than with our track excerpts simply because more frames are available.

Figures [fig:risampleerror] and [fig:risuppratio] show the quartile statistics of the sample error and suppression ratio with re-injected ground truth speed. We can see how most variants are improved in error spread for the former, and 4 variants are greatly improved for the latter, confirming the sensitivity of the track removal step 3 on the speed estimation. The sample alignment itself is less sensitive to the speed estimation, as seen in [tab:sample_to_risample], which shows the statistics of the difference between sample-alignment error metrics with and without ground truth speed re-injected: only a few variants obtain improvements of 1 or 4 ms.




*[fig:dtw short] DTW distance matrix, alignment path (red), track boundaries and found slope lines on an artificial DJ mix from our dataset*

Some of our experiments with the database tracks also demonstrate the limits of our approach, as can be seen in [fig:dtw short], where the heuristics of choosing the middle of the track as anchor, and the mean slope approach failed to estimate the correct slope for the last track (which was mixed here with double tempo).

## 3.6 Conclusions and Future Work

The presented work is a first step towards providing the missing link in a chain of methods that allow the retrieval of rich data from existing DJ mixes and their source tracks.

An important result is the validation of track suppression as a metric for the accuracy of sample alignment. This metric can be obtained even without ground truth. A massive amount of training data extracted from the vast number of collections of existing mixes could thus be made amenable to research in DJ practices, cultural studies, and automatic mixing methods.

With some refinements, our method could become robust and precise enough to allow the inversion of fading, EQ and other processing. First, the obtained tempo slope could be refined by searching for sample alignment at several points in one source track. This would also extend the applicability of our method to mixes with non-constant tempo curves. Second, a sub-sample search for the best alignment should achieve the neutralisation of phase shifts incurred in the mix production chain.

Extension to mixes with varying tempo within a track is straightforward by extracting a segmented alignment curve from the DTW path. We could also check whether a DTW with relaxed endpoint condition[29] for the beginning and end of a mix could be advantageous.

Furthermore, the close link between alignment, time-scaling, and unmixing in [fig:schema] hints at the possibility of a joint estimation algorithm, maximising the match in the three search spaces simultaneously.

29 Diego Furtado Silva, Gustavo Enrique de Almeida Prado Alves Batista, Eamonn Keogh, et al. On the effect of endpoints on dynamic time warping. In SIGKDD Workshop on Mining and Learning from Time Series, II. Association for Computing Machinery- ACM, 2016.

# 4 In-Store Player Mixing Module

The ISP player module is a GUI-less stand-alone program to be run on the ISP hardware that takes care of beat-synchronous mixing and audio playback. It is controlled only by the ISP scheduler (the control program running on the ISP, responsible for scheduling playlists and communicating with the HearDis! servers and cockpit unit) and thus only dependent on local data and commands. Please see D2.10 *Basic Version of ABC_DJ System Architecture* for details about the system architecture and the role of the ISP scheduler.

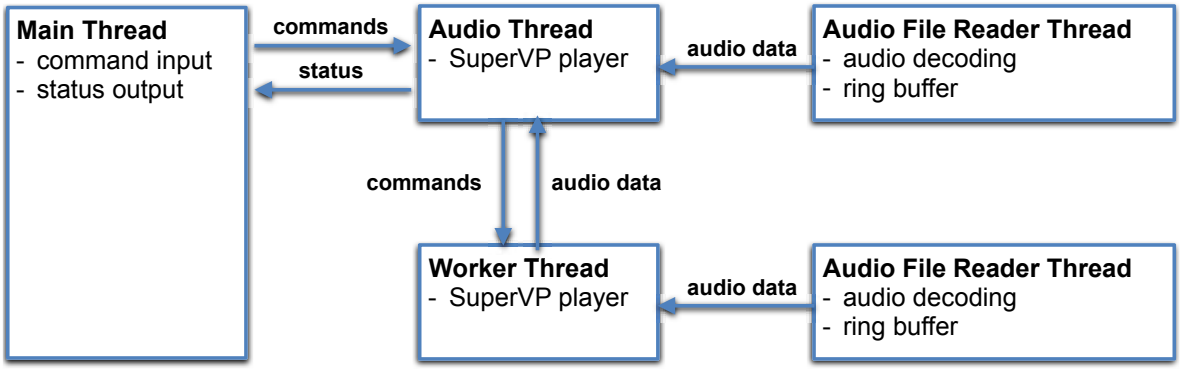
The player communicates via Unix FIFOs (named pipes) with the ISP scheduler. It receives the next track to play from the playlist in JSON format (see below), and outputs status information (about which track is starting/playing/stopping, current bpm and play position) and command acknowledgments. The input format contains all metadata and annotations necessary for beat-synchronous mixing, such as track bpm, beat markers, cue-regions.

The program streams audio data from disk in a background thread and runs two instances of the SuperVP time-stretching engine, the speed factors of which are controlled as described above by the algorithms for play duration compensation and beat-synchronous mixing. Audio output is realised via the cross-platform PortAudio library[30], which maps to CoreAudio for the Mac prototype version, to ALSA for the Linux production version, and to any of the audio APIs (MME, DirectSound, WASAPI, WDM/KS, ASIO) for the Windows demonstrator.

Since with the start of a track, it is already known (from the playlist) at which point in time the next track has to be started, we can work with very large audio buffer sizes. This helps distributing CPU load peaks over time, which results from the fact that, during cross-fade in the cue-regions, two SuperVP time-stretching engines have to run in parallel.

For the project, the SuperVP time-stretching library was ported to the ARMv7l processor architecture of the player hardware (similar to Raspberry Pi model 3, which was the testing environment).

The architecture is multi-threaded in order to distribute the processing load over the 4 processor cores of the player hardware, as depicted in the following diagram:




*[fig:isp-arch] ISP Mixing Module Architecture.*

30 http://portaudio.com

## 4.1 Usage

The invocation and options of the mixing module are given here. It reads play commands in the form of JSON records described in section 4.2 unterhalb from the given input pipe (can be '-' for stdin) and outputs status information in formats described in section 4.4 unterhalb to the output pipe (or "-" for stdout).

A typical invocation in a production context would be:

```
mkfifo /tmp/isp-control

mkfifo /tmp/isp-status

./isp-player -r /data/tracks /tmp/isp-control /tmp/isp-status
```

The full list of options of the mixing module are:

```
USAGE:

   ./isp-player  [-r <path>] [-s <seconds>] [--time <datetime|time>]
                 [--splitchannels] [-n] [-c] [-P] [-p] [-t] [--]
                 [--version] [-h] <infile> <outfile>

Where:

   -r <path>,  --root <path>
     audio file root dir

   -s <seconds>,  --timeout <seconds>
     status reporting interval

   --time <datetime|time>
     for testing: set clock date and time, special value: 'first': use
     first playlist schedule time as clock time

   --splitchannels
     for testing: split voices to left/right channel

   -n,  --nostretch
     mix continuously according to cue-regions and beat-markers, don't
     apply time-stretch

   -c,  --continuous
     mix continuously according to cue-regions, ignoring track start times
     from playlist

   -P,  --prettyevents
     json event status output pretty printed and multi-line, running status
     still one line

   -p,  --pretty
     json status output pretty printed and multi-line

   -t,  --textoutput
     status output in text instead of json
```

```
--,  --ignore_rest
  Ignores the rest of the labeled arguments following this flag.

--version
  Displays version information and exits.

-h,  --help
  Displays usage information and exits.

<infile>
  name of input pipe or '-' for stdin

<outfile>
  name of status output file, pipe, or '-' for stdout


ABC_DJ in-store player audio player and mixer module.

Expects playlist entries in json format on input, outputs status
information in json on output.
```

## 4.2 Command Scheduling

There are two constraints for when certain commands can be sent from the ISP scheduler to the mixing module:

1. After starting the mixing module, the ISP scheduler must wait for it to initialise, after which it will send the “startup” event. Only then can tracks be scheduled for playing.
2. The play command must not be given when the instance of the corresponding SuperVP engine is still busy. In other words, to schedule track *n+1,* the ISP scheduler must wait at least for the end of track *n-1*. A safe possibility is to send the play command for track *n+1* a few seconds before its start according to the playlist.

## 4.3 Input JSON Format

The input format sent by the ISP scheduler contains the scheduling information for a track (start time, file path), and all metadata and annotations necessary for beat-synchronous mixing, such as track bpm, beat markers, cue-regions. Below is an example for one track, that will be sent to the ISP mixing module’s control FIFO. All fields used by the module are marked in bold.

For testing purposes, the ISP player also accepts the command “play <filepath>” to mix a file immediately into the running stream with a default fade-in and fade-out of 10s.

```
{
    "filepath": "/data/tracks/02_David Bowie_Lazarus.wav",
    "created" : "2018-11-29T15:45:39.000+0000",
    "startTime": "2017-09-14T00:00:00",
    "endTime": "2017-09-14T00:02:04.627",
    "manualMetadata" : {
        "timeReference" : "2010",
        "type" : "music"
    },
```

```
    "id" : "e9aec19b-43d5-4ca5-a3c0-6f58041922e4",
    "embeddedMetadata" : {
        "length" : 355.58,
        "tracknumber" : 2,
        "label" : "Columbia",
        "album" : "Blackstar",
        "heardis_id" : "HD_2016_1663680881",
        "codec" : "mp3",
        "createDate" : "2016",
        "artist" : "David Bowie",
        "resolution" : 320,
        "year" : 2016,
        "sampleRate" : 44100,
        "title" : "Lazarus"
    },
    "automaticMetadata" : {
        "mono" : true,
        "subStyles" : [
            "indie-pop",
            "aor",
            "fusion-jazz"
        ],
        "genre" : "blues",
        "instrumentation" : "electric-guitar",
        "lowQuality" : false,
        "vocals" : true,
        "mainStyle" : "indie-dance",
        "bpm" : 130,
        "cuePoints" : [
            {
                "id" : "15",
                "type" : "cue-in",
                "description" : "intro",
                "startTime" : 18.63,
                "source" : "ircam",
                "length" : 10
            },
            {
                "type" : "cue-out",
                "id" : "15",
                "description" : "outro-loudenough",
                "source" : "ircam",
                "startTime" : 336.14,
                "length" : 10
            }
        ],
        "loudness" : 4.32,
        "beats" : [
            0.24,
            0.71,
            1.17,
            1.62,
            2.08,
            2.54,
            3,
            3.46,
```

```
            ...
        ]
    }
}
```

## 4.4 Output JSON Format

The player outputs the following status information in JSON format:

- The current player state is sent as a heartbeat every few seconds (configurable), so that the ISP scheduler can always monitor the current player state and transmit it to the cockpit unit. The state contains the following information:
  - currently playing track names and bpms
  - position within playing tracks
  - current and next track start time
  - current global volume
- Event information is sent as direct feedback after control input is received:
  - player startup feedback
  - acknowledgement of scheduling of next track with the scheduled start time
  - events when a track starts or stops playing
- Error messages alert the ISP scheduler about unforeseen conditions (missing files, audio I/O errors, buffer underruns, invalid metadata, etc.)

In the following we give JSON templates (with the most important fields) and examples for the three types of messages.

### Status Message

The status message contains the current time and the data for the currently playing tracks:

```
{
  "messagetype": "status",
  "time":        <time since player start in seconds>,
  "timestamp" : "<clock time>",
  "tracks":
  [
    {
      "trackname":  "<track name>",
      "position":   <position in track in s>,
      "starttime":  <player time of track start>,
      "stoptime":   <player time of track end>,
      "trackbpm":   <original BPM>,
      "currentbpm": <current BPM>,
      "volume" :    <current track volume>,
      "voice" :     <voice index 0 or 1>
    },
    { /* possibly second playing track */ }
  ]
  "volume": <current volume factor>
}
```

Below is a full example of a status message when two tracks are playing:


```
{
  "messagetype" : "status",
  "numplaying" : 2,
  "time" : 13.838918648019899,
  "timestamp" : "2018-12-20T13:35:33",
  "tracks" :
  [
    {
      "bpmend" : 110.0,
      "bpmstart" : 110.0,
      "cueinend" : 4.8636363636363633,
      "cueinstart" : 0.5,
      "cueoutend" : 15.77209090909091,
      "cueoutstart" : 11.409090909090908,
      "currentbpm" : 110.0,
      "currentspeed" : 1.0,
      "duration" : 60.5,
      "voice" : 0,
      "position" : 13.507528344671202,
      "starttime" : 0.83648833801271394,
      "status" : "fadeout",
      "trackbpm" : 110.0,
      "trackname" : "clicktrack-110bpm+0.5s.aiff",
      "volume" : 0.50033760070800781
    },
    {
      "bpmend" : 120.0,
      "bpmstart" : 120.0,
      "cueinend" : 4.5,
      "cueinstart" : 0.5,
      "cueoutend" : 14.5,
      "cueoutstart" : 10.5,
      "currentbpm" : 110.00000953674316,
      "currentspeed" : 0.91666674613952637,
      "duration" : 60.5,
      "voice" : 1,
      "position" : 2.4665532879818595,
      "starttime" : 11.745579247103624,
      "status" : "fadein",
      "trackbpm" : 120.0,
      "trackname" : "clicktrack-120bpm+0.5s.aiff",
      "volume" : 0.48910710215568542
    }
  ]
}
```


### Event Message

The event feedback message can contain optional track data:


```
{
  "messagetype": "event",
  "time":        <time since player start in seconds>,
  "timestamp" : "<clock time>",
```

```
  "eventtype": "startup|schedule|trackstart|trackend",
  "tracks":
  [
    {
      "trackname":  "<track name>",
      "position":   <position in track in s>,
      "startedat":  "<clock time of scheduled track start>",
      "stoptime":   "<clock time of scheduled track end>",
      "trackbpm":   <original BPM>
      "currentbpm": <current BPM>
    }
  ]
}
```

Here is an example of the event feedback to the play command:

```
{
  "eventtype" : "play",
  "messagetype" : "event",
  "time" : 1.8580194360110909,
  "timestamp" : "2018-12-20T13:35:21",
  "tracks" :
  [
    {
      "bpmend" : 120.0,
      "bpmstart" : 120.0,
      "cueinend" : 4.5,
      "cueinstart" : 0.5,
      "cueoutend" : 14.5,
      "cueoutstart" : 10.5,
      "currentbpm" : 120.0,
      "currentspeed" : 1.0,
      "duration" : 60.5,
      "voice" : -1,
      "position" : 0.0,
      "starttime" : 11.745579247103624,
      "status" : "uninitialized",
      "trackbpm" : 120.0,
      "trackname" : "clicktrack-120bpm+0.5s.aiff",
      "volume" : 0.0
    }
  ]
}
```

### Error Message

```
{
  "messagetype":  "error",
  "time":        <time since player start in seconds>,
  "timestamp" : "<clock time>",
  "errormessage": "message",
}
```

Two examples of error messages:


```
{
  "errormessage" : "ioerror",
  "messagetype" : "error",
  "time" : 123.456,
  "timestamp" : "2018-12-20T13:44:08",
  "tracks" : null
}

{
  "errormessage" : "no embeddedMetadata in json for clicktrack-120bpm.aiff",
  "messagetype" : "error",
  "time" : 0.72091909596929327,
  "timestamp" : "2018-12-20T15:34:40",
  "tracks" : null
}
```

# 5 Summary and Conclusions

This deliverable describes research results, datasets, and software delivered to the project partners pertaining to automatic DJ mixing on the ABC_DJ in-store player. We still expect some further adaptation of the software (in WP 5 Integration) after more experience with real-world data and playlists.

We had initially foreseen that the publication of the ISMIR 2017 position paper *Towards Ground Truth Extraction of DJ Mixes*, and the ISMIR 2018 *Unmixdb* open dataset for DJ mix reverse engineering would improve the repeatability of DJ-related MIR and would help to establish it as a research field in its own right. This hope seems to have been fulfilled, since we were contacted by a young researcher whose master's thesis[31] was, in his own words, "basically building upon the concepts in your paper *Towards Ground Truth Extraction of DJ Mixes*". It treats the extraction of ground truth from DJ Mixes in order to make them amenable to machine learning to advance automatic methods for DJ support. He published software and a database of DJ mixes hand-crafted in the Ableton *Live* software from which he can extract ground truth annotations to test his methods. This database is perfectly complementary to our automatically generated *Unmixdb* dataset.

[31] Werthen-Brabants, Lorin, and Tijl De Bie. *Ground Truth Extraction & Transition Analysis of DJ Mixes*. Master of Science in Computer Science Engineering, August 2018.