Automatic Control of a Digital Reverberation Effect using Hybrid Models

Emmanouil Theofanis Chourdakis[1] and Joshua D. Reiss[1]

[1]*Queen Mary University of London, Mile End Road, London E14NS, United Kingdom*

Correspondence should be addressed to Emmanouil Theofanis Chourdakis (e.t.chourdakis@qmul.ac.uk)

ABSTRACT

Adaptive Digital Audio Effects are sound transformations controlled by features extracted from the sound itself. Artificial reverberation is used by sound engineers in the mixing process for a variety of technical and artistic reasons, including to give the perception that it was captured in a closed space. We propose a design of an adaptive digital audio effect for artificial reverberation that allows it to learn from the user in a supervised way. We perform feature selection and dimensionality reduction on features extracted from our training data set. Then a user provides examples of reverberation parameters for the training data. Finally, we train a set of classifiers and compare them using 10-fold cross validation to compare classification success ratios and mean squared errors. Tracks from the Open Multitrack Testbed are used in order to train and test our models.

1. INTRODUCTION

Digital Audio Effects (DAFx) are transformations on an audio signal, or a set of audio signals, where the transformation depends on a set of control parameters. In general, users of DAFx control these parameters themselves and they tend to change these parameters over time based on how the audio sounds. They assign specific audio features (or their changes) to specific parameters (or changes). Unknowingly, they are doing a form of classification where the samples are features of the audio, and the classes are parameter sets. Our goal is to simulate this process using a supervised learning approach to train classifiers so that they automatically assign effect parameter sets to audio features. This way, we can train our reverberation effect to decide how to choose its parameters based just on the observed audio.

In order to achieve this, we perform factor analysis to select the best features from a 31-dimensional feature space from 8 features found in the literature. Pre-processing is applied on the resulting features as well as on the parameters which we cluster into sets using k-means. We then compare 4 different classifiers on the classification task where our samples are vectors of audio features and classes are the parameter-set clusters. The training data consists of the control parameters provided by the user with a simple interface that allows her to control a simple reverberation effect. Testing is done using cross-validation.

2. PREVIOUS WORK

There has been a lot of research in Adaptive Digital Audio Effect for automatic multitrack mixing but in almost all cases they focus on achieving a pre-specified goal. Parameter automation and intelligent control have been applied to many of the most popular audio effects (e.g. gain and faders [1], equalization [2], panning [3] and dynamic range compression [4]), but to the best of the authors' knowledge it has not been attempted on artificial reverberation. Furthermore, all of the above mentioned approaches except [1] which use Linear Dynamical Systems to estimate mixing weight coefficients, use fixed rules, rather than arbitrary rules that are learned from training data. On the other hand, to the knowledge of the authors, there are no published works on Automatic Application of Reverberation. Important work however, is found in [5] where the authors try to assign descriptive terms to reverberation parameters. Similar work can be found in [6] where they use a Reverberation Effect among others for their S.A.F.E semantic audio project which allows users to assign high level descriptive terms to low level audio feature changes that are caused by effect parameter changes. In a similar fashion, [7] create a map of high level descriptive terms that correspond to

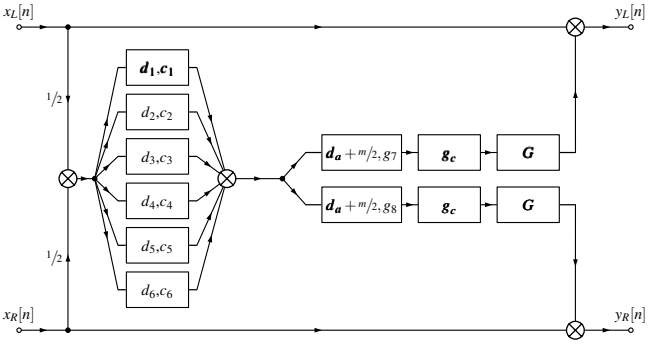

Fig. 1: Rafii & Pardo's Algorithmic Reverberator. Filter boxes are represented by their control parameters. d_ν, g_ν for $\nu = 1 \ldots 6$ represent Comb Filter delays and gains respectively, d_a all-pass filter delays, g_c low-pass filter gain and G is a dry/wet mixer gain. User can control the parameters shown in bold.

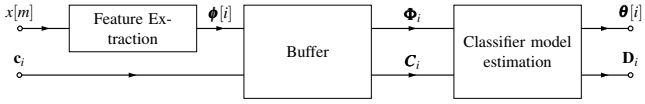

Fig. 2: Training of classifier models

low level reverb parameters. Relevant work in [8] performs classification for drum sounds in order to control effect parameters, but still relies on fixed rules.

3. EFFECT ARCHITECTURE

Our proposed design uses the traditional adaptive DAFX design [9] limited to one track and can be seen in detail in Figure 3. It consists of an artificial reverberator effect where the values of the parameters are decided by a classifier model. The classifier model can be trained online or off-line. The architecture of the model training process can be seen in Figure 2 where $\boldsymbol{\phi}_i$ is the feature vector of the i-th frame, $\boldsymbol{\Phi}_i$ is a matrix of features which consists of the vertical concatenation of the feature vectors (as row vectors), from frame 1 to frame i. In a similar fashion, $\boldsymbol{C}_i$ is a matrix which consists of control parameters. $\boldsymbol{\theta}$ is the classifier parameter vector, $\boldsymbol{D}$ a dictionary that maps class labels to reverberator parameter sets, and $\boldsymbol{c'}$ are the selected effect parameters.

Note that, several features require the accumulation of a number of samples in a buffer (i.e. spectral features) to be computed. In such cases, latency equal to size of the buffer $\times$ the size of the frame is introduced. Similarly, some classifier models require the accumulation

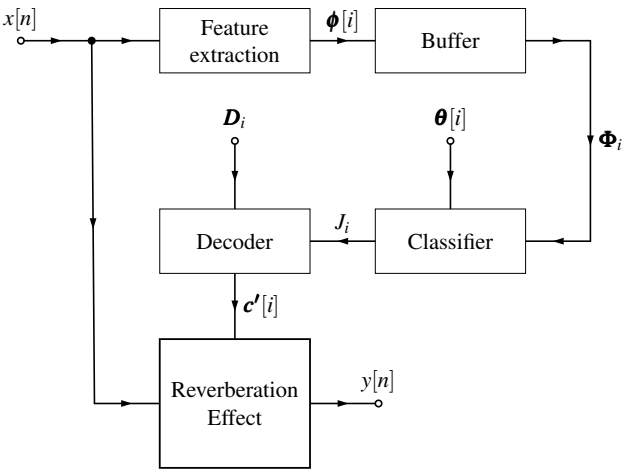

Fig. 3: Reverb application

of several values before being able to make a decision and therefore introduce latency equal to number of previous values $\times$ buffer size $\times$ size of each frame. Consequently, our architecture, although implementable in real time, can introduce latency that depends on the features chosen, as well as the models used. One therefore should be careful in their choice of features and classifier model.

For our reverberation effect we use the algorithmic design described in [5] (Figure 1) because of its simplicity in design and the fact that it can be trained directly from measurements of the reverberation. However, our architecture is not limited to just this particular reverberation model. The parameters of the reverberation and what they affect can be seen in Table 1.

Parameter	Controls	Affects
d_1	Comb filter array	Reverberation Time Echo Density Central Time
d_a	All-pass filter array	Echo Density Central Time
g_1	Comb filter array	Reverberation Time Clarity Central Time
g_c	Low-pass filter array	Spectral Centroid
G	Dry/Wet mix	Reverberation Time Clarity

Table 1: Algorithmic Reverberation Parameters

4. FEATURE EXTRACTION

Application of reverberation to a track can depend on the instrumentation, the type of music and the percussiveness of the track, among others. For our task we use 8 different features. Their names and their role can be seen in Table 2 [10], [11], [12]. The reason for choosing these features is because they have been used extensively in the literature for classification of instruments based on the above characteristics.

Before extracting the features from our audio, we first split the audio into 23ms frames (1024 samples at 44.1Hz, commonly used in the literature for audio-based machine-learning tasks) using the onset-based audio segmentation method described in [13] which is based on the *Spectral Contrast* feature. The reason for choosing this kind of segmentation, as shown in the same paper, is that it appears to give higher classification accuracies for at least the music-genre classification task. We then concatenate our features into a 31 dimensional vector[1] for each frame. Next, we use Principal Component Analysis [14] to filter out nonseparable or noisy features and reduce our feature vectors' dimensionality. We use Horn's Parallel Analysis [15], [16] to find the optimal number of Principal Components to keep.

Feature	**Used in**
ZeroCrossingRate	Instrument Identification Source Identification
13 *MFCCs*	Instrument Identification Genre Classification
12 *Spectral Contrast Coefficients*	Instrument Identification Genre Classification
Root Mean Square	Instrument Identification Voice/Music Discrimination Audio Activity Detection
Crest Factor	Instrument Identification
Spectral Centroid	Instrument Identification Genre Classification
Spectral Roll-off	Instrument Identification Genre Classification
Spectral Flux	Instrument Identification

Table 2: Used features and their usage in the literature.

[1] *MFCCs* and *Spectral Contrast* features have 13 and 12 dimensions respectively.

5. CLASSIFICATION

We use classification on the audio features in order to control the values of the reverberation effect parameters. Given audio segments with applied reverb we do the following:

1. Assign a different label for every different set of reverb parameters assigned. The labels will constitute our class labels. Keep a dictionary structure comprised of the parameter sets and the labels to which they correspond.
2. Segment the audio segments to frames and calculate a feature vector for each frame.
3. Train a classifier model that classifies the features extracted to the labels given above.

For the application of reverberation:

1. Segment the audio, to which we want to apply reverb, to frames and calculate a feature vector for each frame.
2. Classify the new features to the class labels found in the training phase.
3. Use the dictionary structure derived in the training phase to change from class labels to reverberation effect parameters.

More specifically, for the training phase, we consider the vectors:

$$\mathbf{c}_i = \left[d'_{1i}\ d'_{ai}\ g'_{1i}\ g'_{ci}\ G'_i\right]^T$$
$$\boldsymbol{\phi}_i = \left[\phi_{i1}\ \phi_{i2}\ \ldots\ \phi_{iM}\right]^T$$

The elements of $\boldsymbol{c}_i$ are the values of the parameters shown in Table 1 normalized to $(0,1)$, at frame i. Correspondingly, the elements of $\boldsymbol{\phi}_i$ are values of features of Table 2, normalized again to $(0,1)$ at frame i. We also consider the (unique) sets:

$$C = \{\mathbf{c}_i : \forall i\}$$

then $|C|$ is the number of classes. We then define the variable:

$$J_i = \sum_{k=1}^{|C|} k[\mathbf{c}_i == C_k], \forall i = 1\ldots M, k = 1\ldots N$$

We will call J our *parameters label* variable. Consistently, we will call A with values $\mathbf{a}_{1...I}$ our *parameters* variable and F with values $\mathbf{f}_{1...I}$ our *features* variable.

We compare 5 different classifiers: *Gaussian Naive Bayes Classifier* [17], *All-vs-all Linear Support Vector Machine (SVM) Classifier* [18], *Hidden Markov Model Maximum A-posteriori Classifier* [17], a hybrid *HMM* classifier with observations taken from a set of SVMs [19] and one we will call the *Sinkhole approach* classifier, that uses the classification results of all of the above and a voting scheme. The last three take into consideration past time events while the first two consider just the current frame. Below we give a small description of each.

5.1. Gaussian Naive Bayes Classifier

Gaussian Naive Bayes Classifier (In short *GNBC*) is a version of the naive Bayes classifier for continuous variables. Like its non-continuous counterpart, it is based on the Bayes rule:

$$p(X|O) = \frac{p(X)p(O|X)}{p(O)}$$

where X and O are random variables and $O \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ where $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ are the model's parameters. If we substitute O with F_i and X with J_i, then the probability we have a specific set of parameters is given by Equation 1.

Given a set of feature vectors $\{\boldsymbol{\phi}_i : i = 1 \dots I\}$, J_i can be calculated using the *Maximum A-Posteriori* Decision rule:

$$J_i = \underset{k=1...|C|}{\operatorname{argmax}} p(C_k) \prod_{j=1}^{I} p(\boldsymbol{\phi}_j | C_k)$$

Inversely, given a set of features ϕ_i and their corresponding labels J_i, we can train the Gaussian Naive Bayes Classifier (which means to estimate the parameters $\boldsymbol{\mu}$, $\boldsymbol{\Sigma}$) using *Maximum likelihood Estimation*.

5.2. AVA Linear Support Vector Machine Classifier

Support Vector Machines (SVMs) are *Maximum Margin* classifiers. Their goal is to find a boundary that discriminates the data points and retains the maximum distance from the closest ones to that boundary (the *margin*). Given a set of feature vectors $\{\boldsymbol{\phi}_i : i = 1 \dots I\}$ the decision boundary of a Linear SVM that classifies these vectors into two classes $\{+, -\}$ is given by:

$$y(\boldsymbol{\phi}_i) = \mathbf{w}^{\mathbf{T}} \boldsymbol{\phi}_i + b$$

and the decision rule is:

$$\boldsymbol{\phi}_i \text{ is classified as: } \begin{cases} + & \text{if } y(\boldsymbol{\phi}_i) \geq 0 \\ - & \text{if otherwise} \end{cases}$$

Training of a Linear SVM is an optimization problem which can be solved using various methods [20].

Extending the SVMs for the multi-class case using binary classifiers can be done in two ways: *One-vs-All* and *All-vs-All* classes (abbreviated *OVA* and *AVA*) [18]. For the OVA case, we find a decision boundary that discriminates between a class k and the other $K-1$ classes, for every k. In the AVA case, we find the decision boundaries for every pair (k,l) of K classes. In this work we (arbitrarily) choose to do AVA classification.

5.3. Hidden Markov Model Maximum Likelihood Classifier

Hidden Markov Models (HMMs) can be considered as *finite state automata* where each state can only be indirectly observed. States are considered hidden variables and each state is assigned one observation variable. First-Order Hidden Markov Models are HMMs where the transition in one state depends only on the previous state.

An HMM of K states and Gaussian emissions is formally given as the tuple:

$$\theta = \langle \boldsymbol{A}, \boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\Pi} \rangle$$

where $\boldsymbol{A}$ is the matrix of transition probabilities between states, $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ are the mean vector and covariance matrix of the Gaussian PDF of the observations, and $\boldsymbol{\Pi}$ is the matrix of the states' prior probabilities.

For our classifier we train $|C|$ HMMs – one for each class – using feature (observation) sequences of length L. For each feature sequence then that we want to classify, we choose as our class label, the label of the HMM that gives us the *Maximum Likelihood* for that sequence. The decision rule is therefore given by:

$$J_i = \underset{k=1...K}{\operatorname{argmax}} \mathscr{L}(\boldsymbol{\phi}_{i-L+1:i}, x_{i-L+1:i} | \theta_k)$$

where θ_k are the parameters of the HMMs for the k-th class, x_i is the hidden state at the i-th moment, $\boldsymbol{\phi}_i$ is the

corresponding emission and $\mathscr{L}(\boldsymbol{e},x)$ denotes the likelihood of the emission sequence $\boldsymbol{e}$ given the state sequence x. Likelihood $\mathscr{L}$ is calculated using the FORWARD algorithm [17]. Each HMM in our case is trained using the VITERBI algorithm [17]. The length of the sequence for each class is found by doing cross-validation on the training set and choosing the length of the chain that gives the best classification accuracy ratios.

5.4. Hidden Markov Models with SVM emissions

The Gaussian HMM classifiers described above work sufficiently, but they rely on Gaussian probability density functions (PDF) for their emissions which cannot discriminate really well. However, SVMs can learn to discriminate well even with very few samples. We would like to combine the discriminating power of the SVMs with the sequential nature of the HMMs but this is not possible directly because SVMs do not provide emission PDFs. Fortunately, we can use some tricks to derive a PDF.

Using Platt's method [19] and following the work done in [21] for our AVA Classifier described in Section 5.2 we have for a class C_k and a sample feature vector $\boldsymbol{\phi}_i$ [21]:

$$p(C_k|\boldsymbol{\phi}_i) = 1/\left[\sum_{l=1,l\neq k}^{K} \frac{1}{\mu_{kl}} - (K-2)\right]$$

where μ_{kl} is the probability that the class is either C_k or C_l, that is:

$$\mu_{kl} = p(C_k \vee C_l|\boldsymbol{\phi}_i)$$

If we regard the class of f_i our HMM's state, then we can compute its emission probabilities by using the Bayes rule:

$$p(\boldsymbol{\phi}_i|C_k) = \alpha \frac{p(C_k|\boldsymbol{\phi}_i)}{p(C_k)}$$

where $p(C_k)$ can be estimated by counting occurrences of C_k in our data and α is the normalization factor. Since our model's states are the same as our classes, the process of classification reduces to predicting the hidden sequence state of our HMM using the VITERBI algorithm.

Training of the HMM in that case is achieved by first training the SVMs that will provide the emission probabilities, then normally training the HMM using the VITERBI or the BAUM-WELCH (FORWARD-BACKWARD) [17] algorithm.

5.5. Sinkhole Approach Classifier

The classifiers in the sections above will perform differently depending of the nature of our features. The *Gaussian*-based classifiers (*GNB*, *HMM*) will perform better when our features are distributed around a Gaussian bell, the *Support Vector Machine*-based classifiers (*AVA*, *HMM/SVM*) will perform better when our features can be discriminated by using straight lines. Similarly, the *Hidden Markov Model*-based classifiers (*HMM*, *HMM/SVM*) will exploit the sequential nature of our features. In some cases, one of the classifiers above may be more suitable than the other for a set of features. We will exploit this fact and as a final classifier-model approach, we will use what we will call a *Sinkhole* approach. That is, we will do classification on our features using all of the above classifiers and then voting for the result.

The decision process in this case is as follows. For each feature vector $\boldsymbol{\phi}_i$ do:

1. Classify $\boldsymbol{\phi}_i$ using the *GNB*, *AVA*, *HMM*, and *HMM/SVM* classifiers, giving it the labels $J_i^{(\text{GNB})}$, $J_i^{(\text{AVA})}$, $J_i^{(\text{HMM})}$, and $J_i^{(\text{HMM/SVM})}$ respectively. Those will be our *candidate* labels.
2. If a candidate label appears more than the rest, pick that as our final label.
3. When there is a draw, pick the label at random from the candidates that are at draw.

6. TRAINING

Training takes place on excerpts from 254 audio files taken from the Open Multitrack Testbed [22]. First the audio data is segmented into meaningful parts (i.e. song phrases, parts of music, etc.) using a similarity matrix and a novelty function as found in [12]. A user is presented with a simple GUI where she can listen and apply reverb to the extracted parts. The segmented parts are split into frames using the method described in [13] and a tuple of features and parameters are extracted for each frame as described in Section 4. Features are filtered with a low-pass filter and parameter values are compressed using K-MEANS in order to reduce the number of distinct parameter labels. The resulting data set is used to train the models described in Section 5.

7. RESULTS

In order to test our models, we split our data into 6 sets. Every set included 90 audio segments except the last

which included 58. Every segment was a part of a Bass, Keyboards, Vocals, Percussion or Saxophone track. The segments were randomly split into sets. Because the training had been done by the user, the training data includes several labeling errors (that is, we incorrectly applied reverb to some parts) which we automatically detect by fitting a *Minimum Covariance Determinant* [23] and relabeling them by applying an initial classification on the mislabeled features.

We define *classification accuracy* as the normalized (to 1.0) number of times the classifier has been correct on its decisions, and *mean squared error* the average of the squares of the difference between an estimated value of a parameter vector and the true value of that vector.

We tested for classification accuracy and mean squared error as such:

- Split every set into 10 parts.
- Use 9 parts for training and 1 for testing. Do this for every combination of 10 parts.
- Use an increasing number of samples from the training set to train the models, and test them against the full testing set. Store the classification ratio and the mean squared error versus the number of samples used for training.
- Finally, use the averages of all combinations to derive the *Classification Accuracy* and *Mean Squared Error* values.

#	GNB	AVA	HMM	HMM^{SVM}	SINK
1	0.716	0.793	0.703	0.696	0.820
2	0.789	0.789	0.729	0.525	0.793
3	0.769	0.736	0.760	0.537	0.709
4	0.689	0.689	0.552	0.562	0.673
5	0.805	0.772	0.769	0.535	0.777
6	0.796	0.898	0.798	0.563	0.875

Table 3: Average Classification Accuracy rates

Using the full training set, we can see the average *Classification Accuracies* in Table 3, and the average *Mean Squared Errors* in Table 4. A comparison graph for classification accuracy can be seen in Figure 4 and for MSEs in Figure 5. The high classification accuracy result is important because it represents the rate of agreement, between the automatic reverberation effect and the user that it trained it, on the parameters of the reverberation. What is more important though, is having a low mean squared error. Mean squared error effectively measures how far the estimated parameters are from the real ones. This means that while the classification accuracy may be high, so the effect and the users agree most of the time, the differences on the parts they do not agree may be too high for the model to be useful. Therefore, the most useful model is the model with the least mean squared error. In our case, the SVM and the SINK-HOLE approaches perform very similarly both in classification accuracy and mean squared error.

#	GNB	AVA	HMM	HMM^{SVM}	SINK
1	0.015	0.013	0.019	0.010	0.006
2	0.005	0.006	0.009	0.007	0.004
3	0.018	0.020	0.014	0.019	0.019
4	0.010	0.010	0.018	0.010	0.010
5	0.097	0.014	0.013	0.017	0.010
6	0.006	0.003	0.012	0.013	0.003

Table 4: Mean Squared Errors for the normalized parameters.

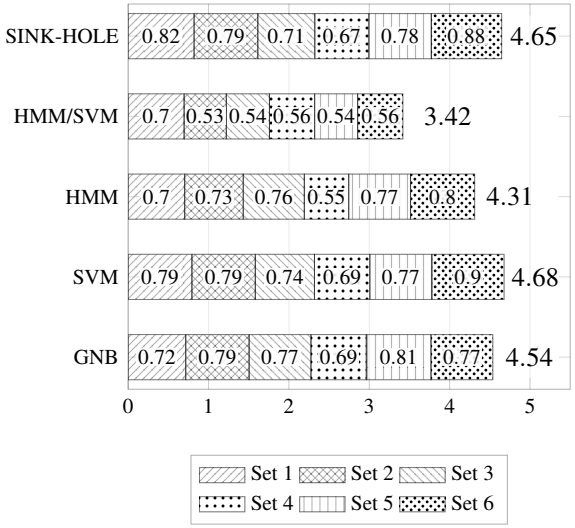

Fig. 4: Classification accuracy scores. Inside the bars is the individual accuracy rate for each model and each dataset. The total score (sum of the ratios) can be seen next to each bar.

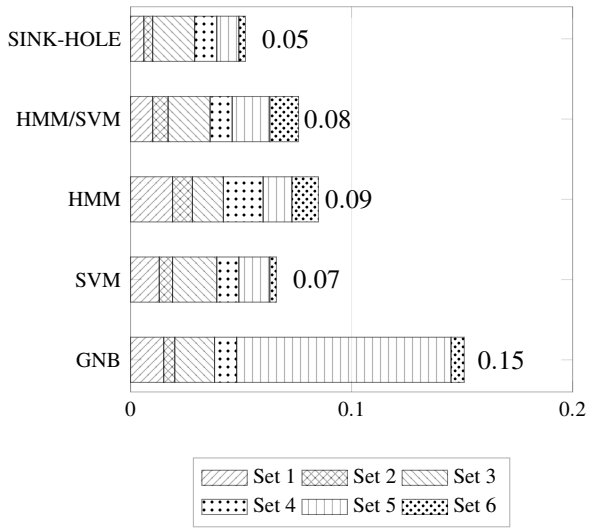

Fig. 5: Mean Squared Error values. The sum of the MSE values for each classifier model can be seen next to each bar.

8. CONCLUSION

From our graphs we can see that for our datasets, the non-sequential models performed better than the sequential counterparts, performing comparably or even worse than the Naive Bayes classifier. This suggests that the Hidden Markov Models failed to capture correctly the temporal progression of our data, possibly suggesting further exploration with different models and configurations. The best model so far seemed to be the All-vs-All Support Vector Machine classifier which performed best regarding Classification Accuracy and Mean Squared Error. In the context of our reverberation effect, this means that our model should provide correct estimates of the parameters 77.9% of the times (the average classification accuracy over all sets).

9. LIMITATIONS/FUTURE WORK

The original Adaptive Digital Audio Effect architecture [9] supports multitrack DAFx, while our method has only been tested for effects applied on a single track. A logical next step would then be to extend our architecture to multitrack audio content. Also, our method is limited to pre-trained sets of parameters, which can be limiting in the introduction of new unexplored audio. [24] describes a way to control continuous control parameters using discrete states using only two parameter states, something which would fit naturally with our approach. Another idea worth exploring is the combination with reverberation effect control using perceptual parameters, as in [5], or descriptive terms, as in [6] and [7]. Especially the two later, since they are dealing by design with sets of parameters, we would expect their method to complement ours very naturally.

10. REFERENCES

[1] J. Scott, M. Prockup, E. M. Schmidt, and Y. E. Kim. Automatic multi-track mixing using linear dynamical systems. In *SMC 2001 - 8th Sound and Music Computing Conference*, 2011.

[2] S. Hafezi and J.D. Reiss. Autonomous multitrack equalization based on masking reduction. *Journal of Audio Engineering Society*, 63(5):312–323, May 2015.

[3] E. P. Gonzalez and J. D. Reiss. A real-time semi-autonomous audio panning system for music mixing. *EURASIP Journal on Advances in Signal Processing*, 2010.

[4] Z. Ma, B. De Man, P. D. Pestana, D. A. A. Black, and J. D. Reiss. Intelligent dynamic range compression. *Journal of the Audio Engineering Society*, 63, 2015.

[5] Z. Rafii and B. Pardo. Learning to control a reverberator using subjective perceptual descriptors. In *10th International Society for Music Information Retrieval Conference (ISMIR 2009)*, 2009.

[6] R. Stables, S. Enderby, De Man B., G. Fazekas, and Reiss J.D. Safe: A system for the extraction and retrieval of semantic audio descriptors. In *15th Int. Society for Music Information Retrieval Conference*, 2014.

[7] P. Seetharaman and B. Pardo. Crowdsourcing a reverberation descriptor map. In *22nd ACM International Conference on Multimedia*, 2014.

[8] J. Scott and Y. E. Kim. Instrument identification informed multi-track mixing. In *14th International Society for Music Information Retrieval Conference*, November 2013.

[9] V. Verfaille, U. Zolzer, and D. Arfib. Adaptive digital audio effects (a-dafx): a new class of sound transformations. *EEE Transactions on Audio, Speech, and Language Processing*, 14:1817–1831, 2006.

[10] G. Peeters. A large set of audio features for sound description (similarity and classification) in the cuidado project. Technical report, IRCAM, 2004.

[11] T. Ganchev, N. Fakotakis, and G. Kokkinakis. Comparative evaluation of various mfcc implementations on the speaker verification task. In *Proceedings of the SPECOM-2005*, 2005.

[12] J. Foote. Automatic audio segmentation using a measure of audio novelty. In *IEEE International Conference on Multimedia and Expo*, volume 1, pages 42 – 455, 2000.

[13] K. West and S. Cox. Finding an optimal segmentation for audio genre classification. In *6th International Conference on Music Information Retrieval*, pages 680–685, 2005.

[14] Y. Lu, I. Cohen, S. Zhou, X, and Q. Tian. Feature selection using principal component analysis. In *Systems Science, Engineering Design and Manufacturing Informatization (ICSEM)*, volume 1, pages 27–30. IEEE, 2007.

[15] J. L. Horn. A rationale and test for the number of factors in factor analysis. *Psychometrika*, 30(2):179–185, June 1965.

[16] A. Dinno. Gently clarifying the application of horns parallel analysis to principal component analysis versus factor analysis. http://doyenne.com/Software/files/PA_for_PCA_vs_FA.pdf, May 15 2014.

[17] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[18] M. Aly. Survey on multiclass classification methods. *Neural networks*, pages 1–9, 2005.

[19] J. Platt. Probabilistic outputs for support vector machines and comparison to regularize likelihood methods. In *Advances in Large Margin Classifiers*, pages 61–74, 2000.

[20] J. Platt. Fast training of support vector machines using sequential minimal optimization. In *Advances in Kernel Methods - Support Vector Learning*. MIT Press, 1998.

[21] S. E. Krger, M. Schaffner, M. Katz, E. Andelic, and A. Wendemuth. Speech recognition with support vector machines in a hybrid system. In *INTERSPEECH 2005 - Eurospeech, 9th European Conference on Speech Communication and Technology*, 2005.

[22] B. De Man, M. Mora-Mcginity, G. Fazekas, and J.D. Reiss. The open multitrack testbed. In *AES 137th Convention, Los Angeles, USA*, 2014.

[23] P.J. Rousseeuw and K. Van Driessen. A fast algorithm for the minimum covariance determinant estimator. *Technometrics*, 41(3):212, 1999.

[24] J. Pazis and M. G. Lagoudakis. Binary action search for learning continuous-action control policies. In *ICML '09 Proceedings of the 26th Annual International Conference on Machine Learning*, 2009.