**EECS 445 Final Report:**

**AutoMixer - Exploring the Mixing Process through Machine Learning**

**Authors**

Loren Wang, Jing Sun, John Bell, Jianqing Huang, Shuang Guan

[{wloren, jings, johnlb, jianhuan, shuangg}@umich.edu](mailto:wloren@umich.edu)

**Abstract**

We propose a system to automate the more common components of the mixing process. By learning common dynamic range compression and EQ settings used to build typical mixes, they can be applied automatically and more of the mix engineer’s time can be spent on more substantive, creative elements of a mix.

**Section 1. Introduction:**

Music is a very popular form of entertainment in our lives, and people have been trying to improve the way we enjoy music for years. Professional DJ’s and audio engineers have the ability to change and shape music to comply with personal taste. However, the average person has little to no experience in the field, yet would like to have the ability to mix music in their own unique way. Our group’s main focus in this project is to explore the mixing process using machine learning methods. In general, we are trying to use examples of professional audio recordings to train an algorithm to mix music in a similar way. This will allow us to create a program that can mix music similar to a professional. Solving this issue will bring about two main benefits. Firstly, the user will be able to listen to all their songs just the way they like it. Secondly, it will introduce a lot of new music into the field, which people worldwide will be able to enjoy.

When creating a recording, individual tracks are first recorded separately. Later, those tracks are manipulated with different processing elements (i.e. dynamic range compression, EQ, reverb, etc.) and combined into 2 tracks which are played back on stereo speakers. This process is called mixing.

Many of these processing elements can be described as multiplying a track by either some time-dependent or frequency-dependent coefficient. However, the exact values of these coefficients are not directly known. The first task is to discover these sets of coefficients by comparing the original tracks with the final, mixed version. These coefficients are known as “ground truth coefficients”. Once the “ground truth” coefficients are known, a linear (or nonlinear) system should be created that can produce a set of coefficient values close to the ground truth values given the same input data. With accurate predicted coefficients, the trained system can automatically mix any new set of tracks.

**Section 2. Related work:**

There are currently only a few attempts to create an automated mixing algorithm. The most prominent of which, conducted by Scott at Drexel University in 2011, uses a system based on a structured audio framework. Their method works in the frequency domain (using STFT (Short Time Fourier Transform)) to find the mixing coefficients. This method, however, results in poor time resolution and the result is very noisy, which they artificially smooth with a Kalman filter. From there, they train a dynamical system to generate the coefficients.

We aim to use a similar approach (find coefficients, train dynamical system), but will improve on it by recognizing that most audio processing is not simultaneously time and frequency dependent. By treating each domain separately, we can get significantly better resolution in both and obtain more robust results. Our approach also adds validation methods to the project. These validation methods can be used to determine how closely we match the actual result, and allow us to tune our hyper parameters.

**Section 3. Proposed method:**

**3.1 Learning Time Domain - Compression**

To find the ground truth mixing coefficients, we first define an RMS transform--where the result x\_rms(n) at any sample n is the Root-Mean-Squared of the previous m samples of the original signal, x(n). That is, x\_rms(n) = RMS( w(tau - n).\*x(tau) ), where w is a rectangular window of width m. The larger tau is, the smoother of a curve we will obtain. This is RMS transformation is applied to every track in the song, allowing us to end with an RMS transform for every instrument (x1\_rms, x2\_rms, etc.).

This RMS signal guards against small timing misalignment between the tracks and makes the result more robust in the presence of noise or secondary modifications made in the mixing process (extra reverb added, for example). Assuming the tracks are uncorrelated, we can say that, on average, a1\*x1\_rms + a2\*x2\_rms + … = t\_rms (where t\_rms is the RMS of the mixed example, a1 and x1\_rms is the gain and RMS of the first signal). Since a is generally time-varying, we use locally weighted linear regression instead of basic linear regression to find **a**(n). We sample both the mixed and original version of each track at regular intervals and interpolate between each result, shown below in Figure 1.

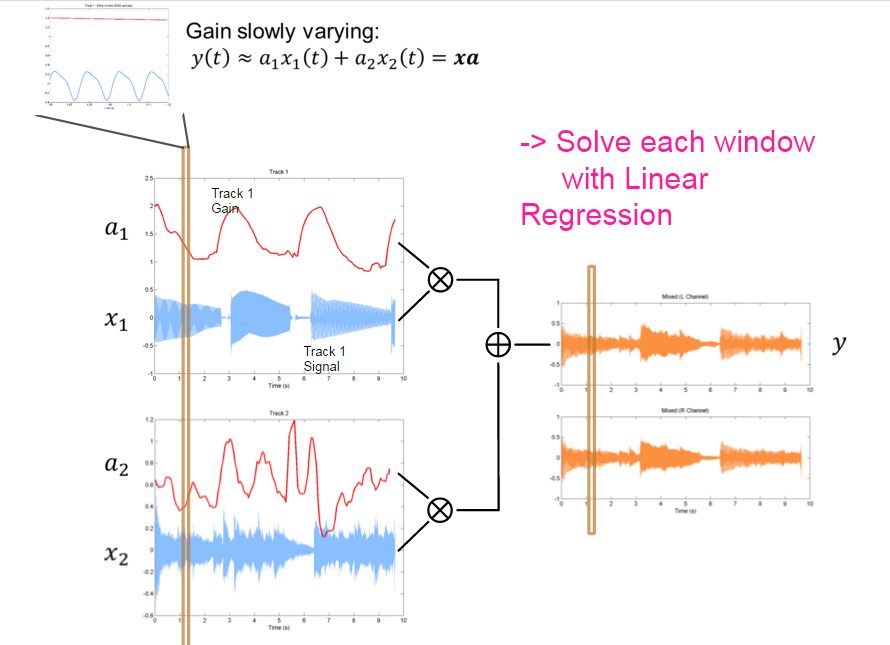


Fig 1. Summing Gains in Tracks 1 and 2

**3.3 Learning Frequency Domain - Equalization**

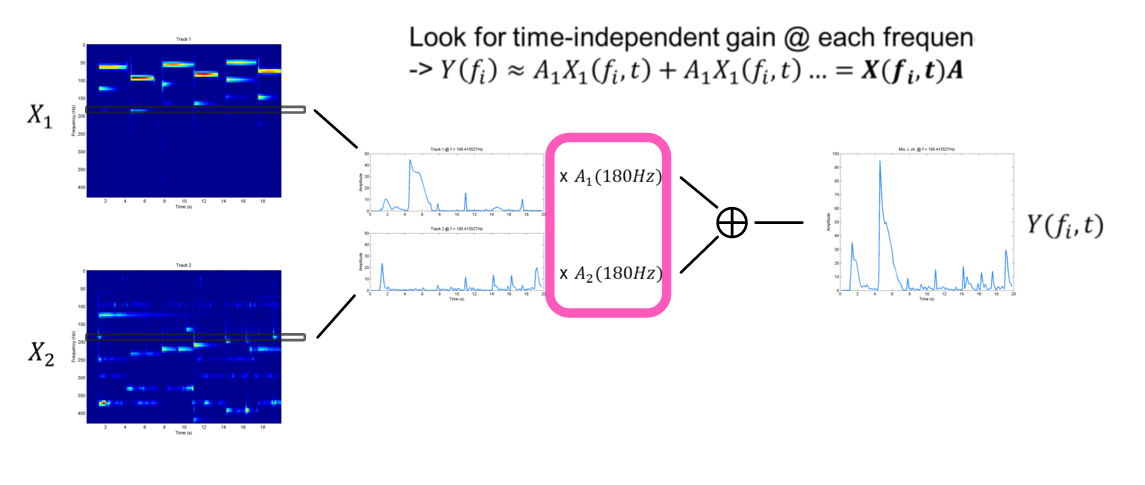
Ground truth frequency coefficients are determined after the gain coefficients have been applied, but before summation. Taking the Short-Time Fourier Transform of the signals gives a picture of how the frequency content varies over time for each track. Then, the energy in each ⅓ octave band is integrated (reproducing human hearing). Since we want the time-invariant coefficients, locally weighted linear regression will again be performed on the integrated dataset. Similar to the time domain gains, we separate each track into windows to compute the gains. However, each window contains a specific frequency range, instead of a specific time period, allowing us to calculate the frequency gains. This process is shown below in figure 2.

Fig 2. Summing up Frequency Domain gains

Once the ground truth has been established, a linear dynamical system will be estimated based on the relationship between the x\_rms signals and ground truth coefficients. This will be accomplished using Matlab’s System Identification Toolbox.

Finally, the resulting systems (one for each example song) can be used as examples to train a generic system that incorporates the important features of the example systems.

A block diagram of this method is shown below.

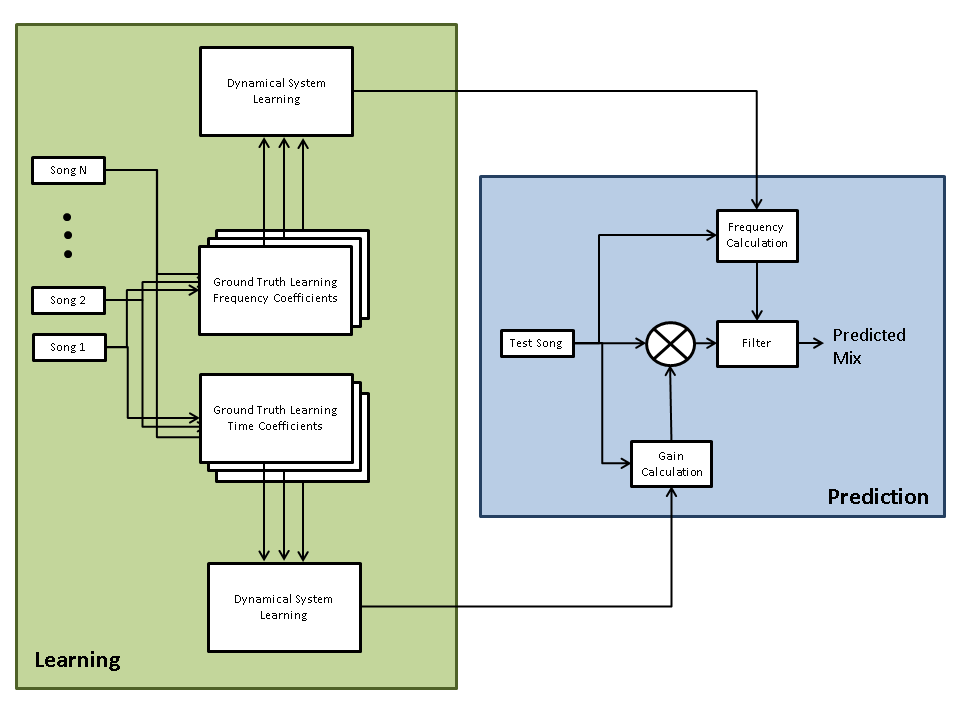


Fig 3. High Level Learning Pipeline

**3.3 Hyperparameter Selection**

Hyperparamters:

* RMS window size
  + Trial and error
  + Smaller window => better time resolution, but less robustness
* Weighted Regression window size
  + Trial and error
  + Larger window => more robustness
* FFT size
  + Fixed: 1024 gives 20Hz frequency precision, which is smaller than smallest integration window
* Frequency Integration Width
  + Fixed: mimics standard filtering capabilities
* Order of Dynamical System
  + Trial and error

**3.3.1 Window energy threshold**

For each track, the signal could be zero during some period. When we are going to find the coefficient for those zero-signal period, the coefficients got from closed form solution could be extremely large which are unreasonable. For those zero-signal period, we choose to adjust the gain to zero manually. In our implementation, we choose to calculate the energy inside a window and set an energy threshold for each window to verify whether there is signal or not.

To find the best threshold for each window, we need to do the validation on our data and then get the best hyper-parameter. First, we will generate some gain for each track manually, like sinusoid signal. Second, apply those gain to each track and sum all those signal we could get a mixed version. Also, we can predict different gain coefficients for each track using our model by adjusting the energy threshold. Finding the distance and correlation between those two output signals can help us to find the best hyper-parameter.

**3.4.2 RMS window size**

**3.5 Proposed Evaluation Method**

First, we will use cross-validation to see whether the song we mix is similar to its testing data, in the feature space sense. We will perform all our comparisons with a defined distance algorithm. This distance algorithm will compare our output with the test data (the mixed track), find the difference between each parameter in our feature space, and assign each with a specific weight. For example, if two of our parameters are amplitude and frequency, it will find the difference between the two tracks in those parameters, manipulate each with a weight, and return an error value. The best mix will be the one which has the smallest error. This evaluation method intends to find and return the parameters that make a good mix and apply them in our program.

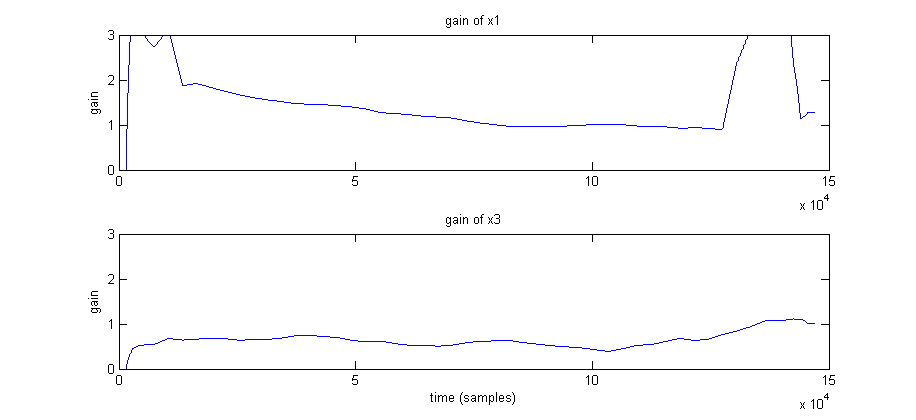
Another similar but more straightforward approach is to take the correlation between our predicted mixed and original mixed signal. The correlation is equal to the inner dot product between two signals, which shows how similar each the two tracks are. The predicted signal will first undergo a time shift to correct for the delay caused by the RMS process. Two dot correlation values are then taken, the original and mixed signal together, and the original signal with itself. We can then define the error function as the distance between these two correlation values, and the best predicted mix will again have the lowest error. To prevent inherent cheating by amplifying the output (a track with a 5x the gain will have 5x correlation), the difference between the RMS amplitudes is added into the error function as a way of regulating high coefficients.

A last evaluation method is for us to simply listen to the produced track and determine whether or not it sounds good. This will allow us to classify the output as either good or bad. We can then feed the result back into our algorithm as a form of supervised learning. A combination of both evaluation methods will incorporated into our final learning algorithm.

Since it is hard to standardize and define how two songs are similar in their style, we may also try to find other methods to evaluate our approach.

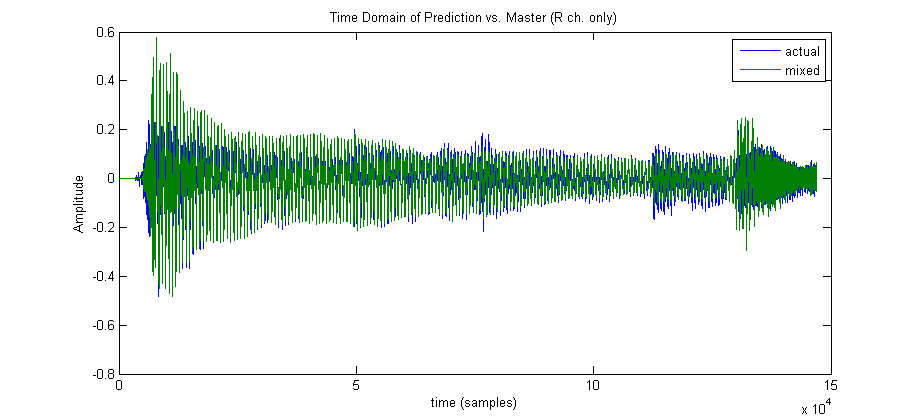
**Section 4. Experimental results and Evaluation:**

**4.1 Data collection and Pre-process**

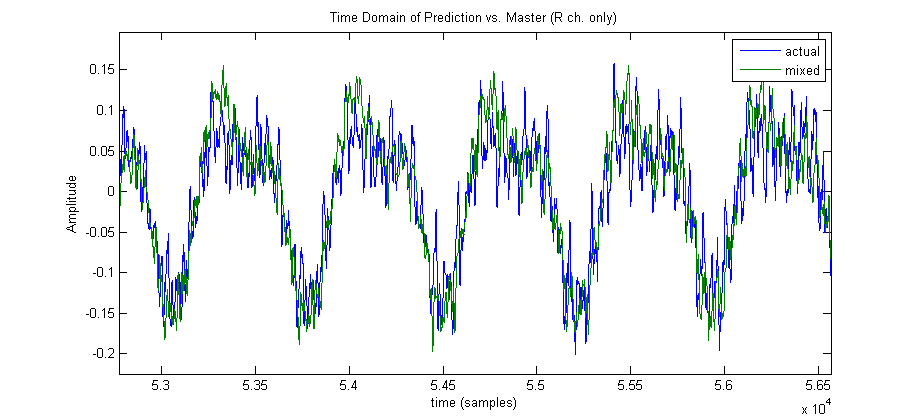


*Extracted gain coefficients for an example with only 2 tracks.*

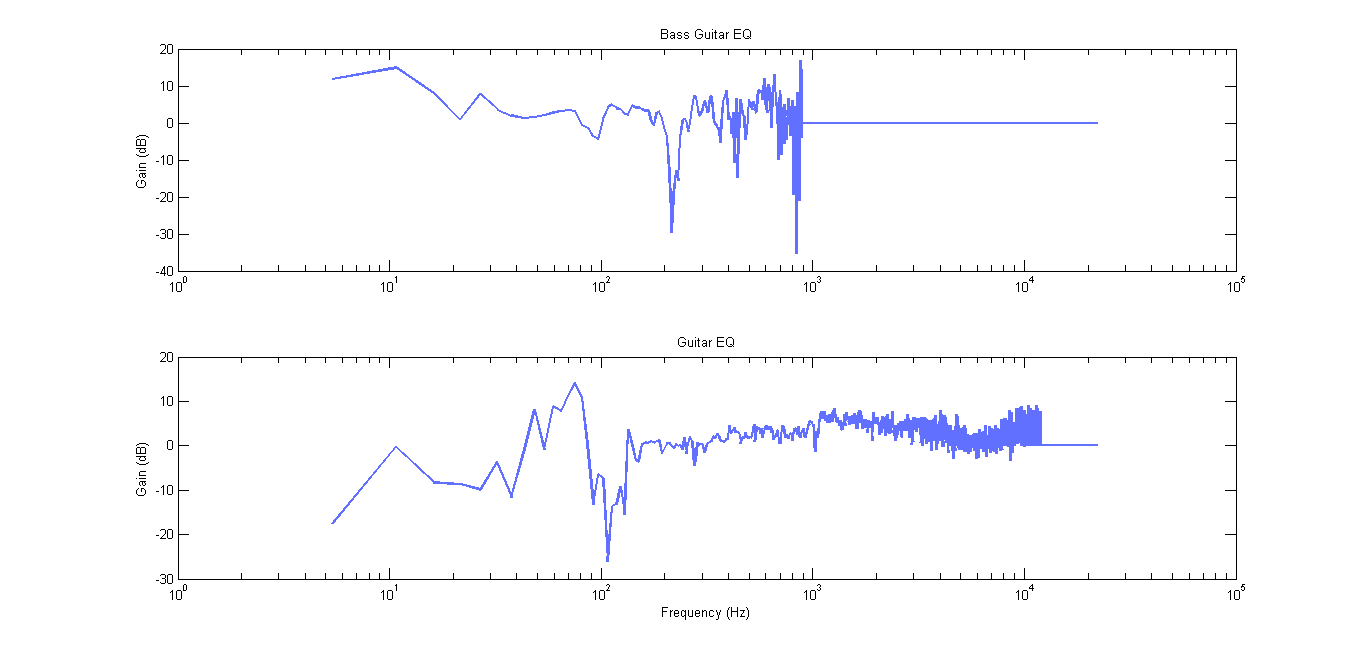
*(note: the large signals at ends of x1 are caused by zero input conditions and will be fixed soon)*



*Comparison of mix using predicted gain values (mixed) and original mix (actual)*



*Same comparison as above, zoomed in*



In our evaluation, we decided to use a mix between the second and third proposed evaluation strategies.

**Section 5. Conclusion:**

Our progress to date has involved a lot of planning, due to the plethora of features involved in creating and mixing music. However, we currently have robust system for finding ground truth coefficients. We now need to focus on generating the dynamical system.

The end goal is to have a system that 1) replicates ground truth reasonably well and 2) consistently ranked higher than unmixed (all mixing coefficients = 1) in blind listening tests.

**Appendix A. Files and Code ?**

Here is a brief overview at the files and functions that we have used in our project:

**References**

[1] J. Scott, M. Prockup, E. M. Schmidt, Y. E. Kim (2011). Automatic Multi-Track Mixing Using Linear Dynamical Systems. *Proceedings of the 8th Sound and Music Computing Conference*, Padova, Italy: SMC.