**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 in a process called mixing where individual tracks are manipulated with different processing elements and combined in certain ways and combined into 2 tracks that played back on stereo speakers. 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. Researchers in the field of auto mixing are trying to find a way to build an automatics models that could learn the unique way the professional mixes music and automatically mixing arbitrary input channels learn. Since we could simply consider the mixing process to be a summation of the linear combination of mixing coefficients (change of gain in operations in dynamic range compression, EQ, reverb, etc.) and tracks, automatics model is achieved by learning from sets of input tracks and associated mixing coefficients. Figuring out reliable mixing coefficients is crucial to build a good automatics model.

Our group’s main focus in this project is to explore this mixing process and find mixing coefficients using supervised learning methods. In general, we are trying to find a reliable model that could figure out mixing coefficients. Reliable mixing coefficients should allow us to reproduce the professional mixing process and mix the raw input tracks similar to a professional. Solving this issue will make it possible to build a good automatics model.

**Section 2. Related work:**

There are currently only a few attempts to find reliable mixing coefficients as well as 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, 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:**

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 mixing coefficients are known as “ground truth coefficients”.

**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.

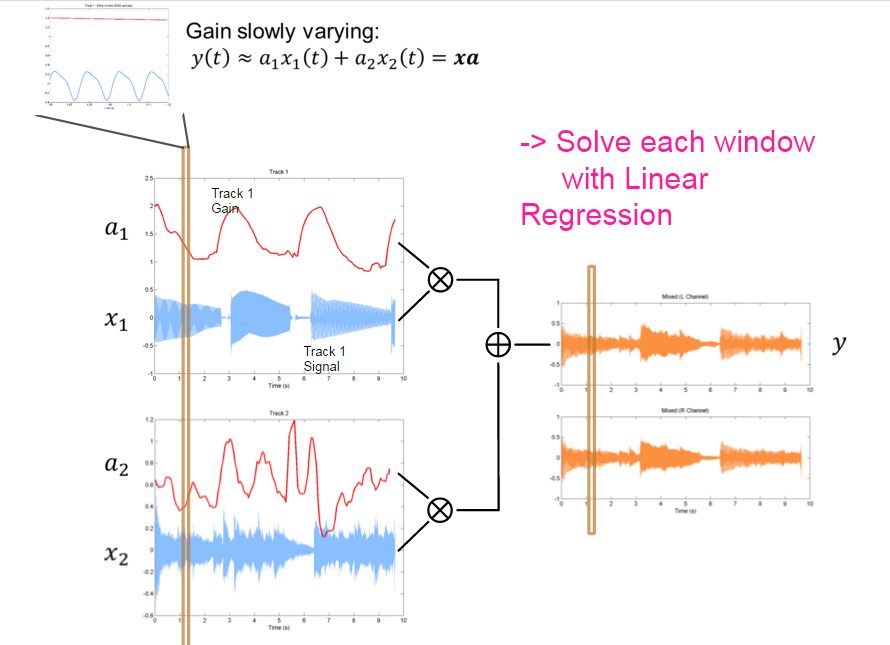


Fig 1. Summing Gains in Tracks 1 and 2

**3.2 Learning Frequency Domain - Equalization**

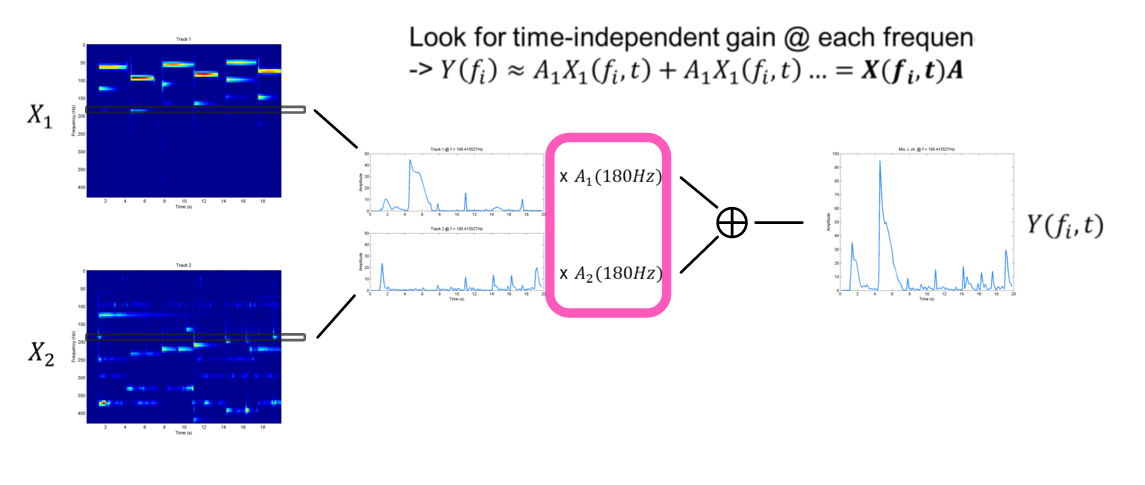
 Ground truth frequency coefficients are determined after the gain coefficients have been applied, but before summation. Taking the Short-Time Fourier Transform of the signals gives a picture of how the frequency content varies over time for each track. Then, the energy in each ⅓ octave band is integrated (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

A block diagram of this method is shown below.

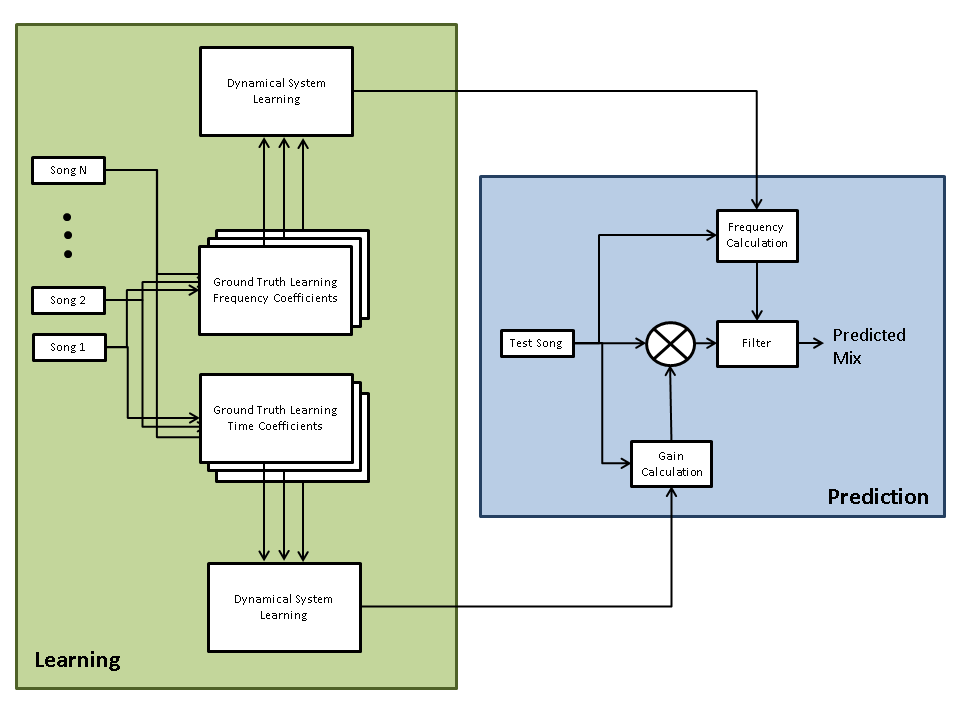


Fig 3. High Level Learning Pipeline

**3.3 Hyper-parameter**

These following hyper-parameters are relevant to find coefficient.

* RMS window size and Regressino Window size

We took Root Mean Square of the signals within certain range, namely, rms window, as one of our feature. And locally weighted linear regression are applied on this feature within a small time range. The larger window, the better time resolution, but the less robustness due to the large variance of signal. We use MSE(Mean Square error) to choose a locally optimal combination of rms window size and regression window size.



* 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. When the threshold becomes large, more signal would be ignored and thus less accurate gain coefficients.

* FFT size

This is a fixed value. We chose 1024 which gives 20Hz frequency precision. It would be smaller than smallest integration window.

* Frequency Integration Width

This is a fixed value which mimics standard filtering capabilities.

**3.4 Proposed Evaluation Method**

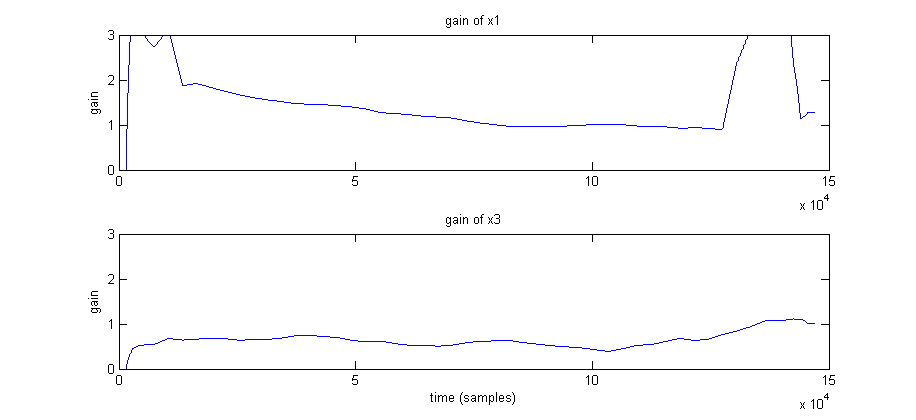
We evaluate the reliability of the mixing coefficients we found in two aspects.

Firstly, we would use MSE to see whether the mixing coefficients we found are similar to the real mixing coefficients. Since in real cases, no real mixing coefficients will be given, we would generate various reasonable mixing coefficient and applied on input tracks to get mixed version as validation data. By doing this, it is possible for us to calculate MSE between our estimated mixing coefficients and real ones to show that the way we found the mixing coefficients are robust.

We would also show that the mixing coefficients can make a good mix. We would 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.

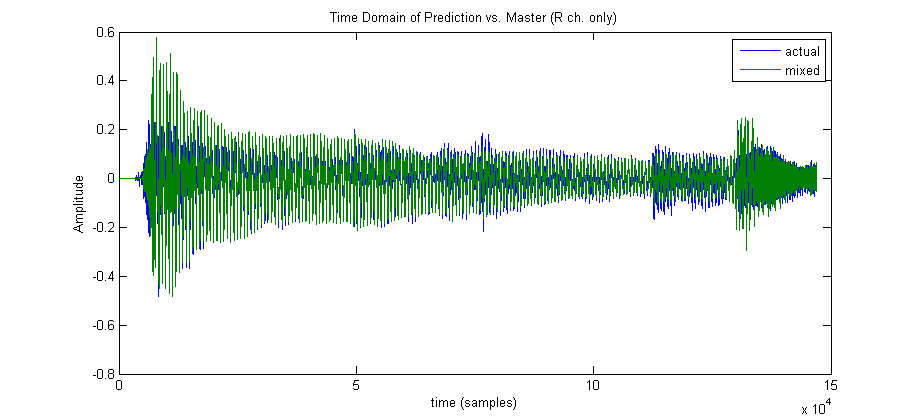
**Section 4. Experimental results and Evaluation:**

**(Need Results)**

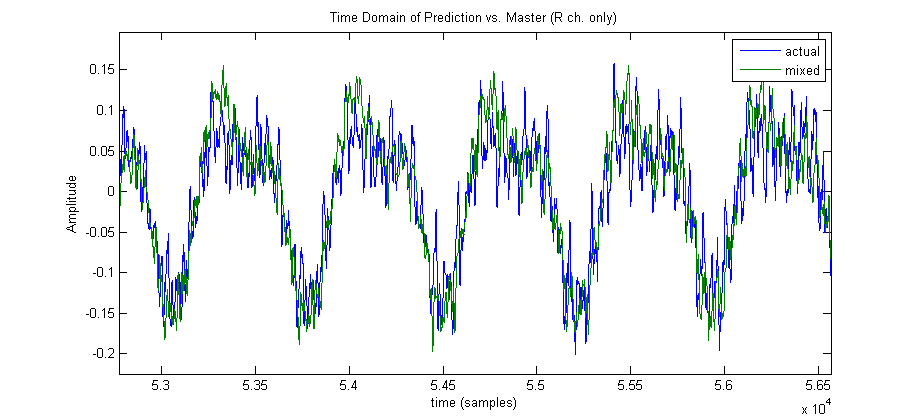


*Extracted gain coefficients for an example with only 2 tracks.*

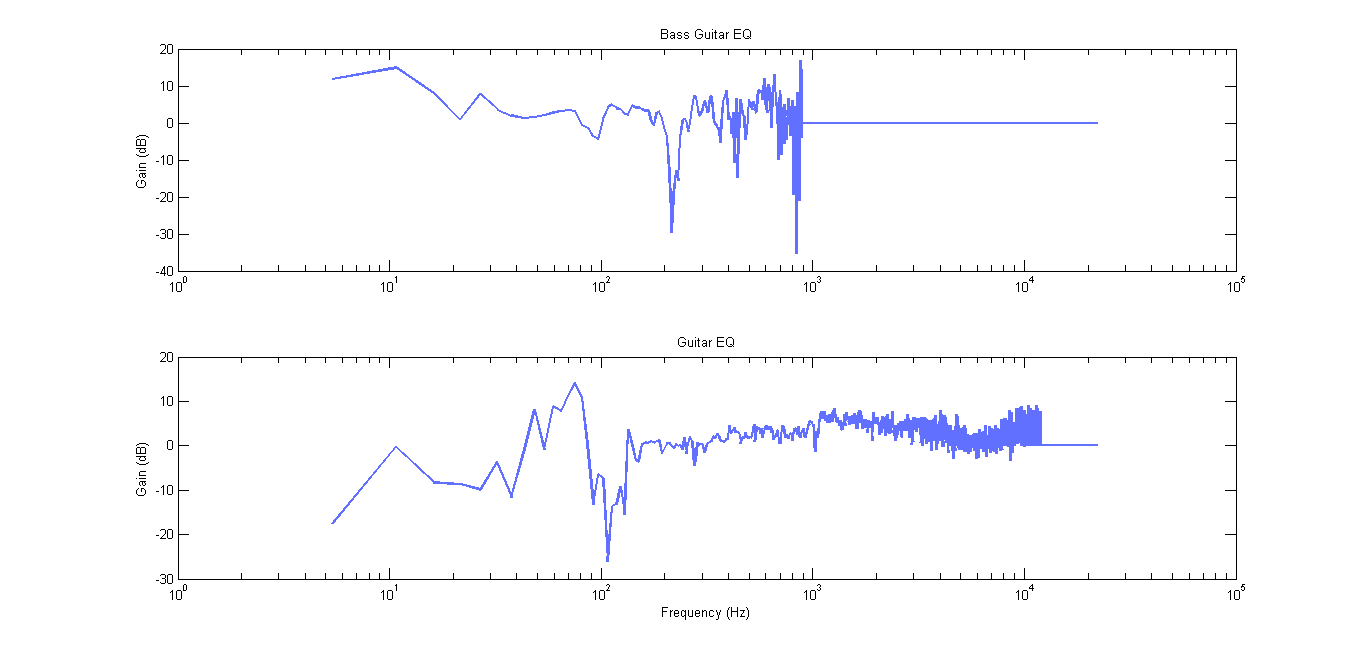
*(note: the large signals at ends of x1 are caused by zero input conditions and will be fixed soon)*



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



*Same comparison as above, zoomed in*



**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.

**Section 5. Future Work:**

One main difficulty we have encounter in this project is data collection and data pre-processing. It is hard for us to get enough unmixed song and mixed song to test our model. Meanwhile, the data size of each tracks are considerable large which takes a long time for us to process the whole songs. Beside, if we want to continue our work to build a model that could auto mix any input tracks. A large number of sets of mixing coefficients are needed for training.

With more time allowed, we would try to solve the problem of the data collection and processing and try to build the real automixer.

**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.