**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. We explore common mixing techniques (such as dynamic range compression and EQ settings) used to build typical mixes, and apply the knowledge to a system that is able to learn these methods and automatically apply them to any song.

**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 automatic models that can learn the unique way the professional mix music and automatically learn to mix arbitrary input channels. Since we can simply consider the mixing process to be a linear combination of tracks and mixing coefficients (change of gain in operations in dynamic range compression, EQ, reverb, etc.), an automatic model is achieved by learning from sets of input tracks and associated mixing coefficients. Figuring out the mixing coefficients is crucial to building a good automatic 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 can figure out the mixing coefficients. Reliable mixing coefficients would 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 currently exists 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 frequency mixing coefficients. This method, however, results in poor time resolution and very noisy results, 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 achieve 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 to achieve the best results.

**Section 3. Proposed method:**

The learning algorithm, our main focus of the project, 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 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**(n) 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.

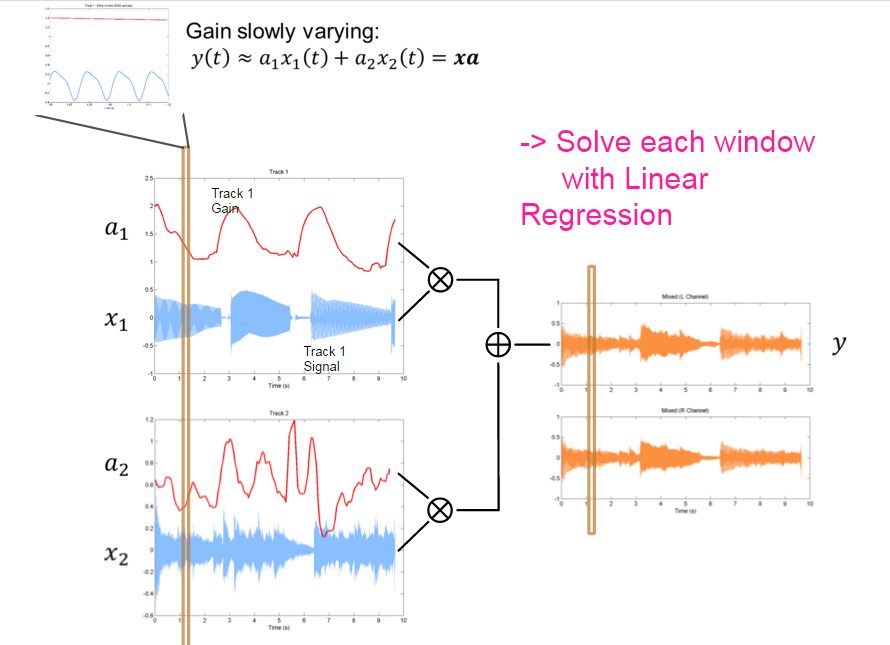


Figure 1. Learning Time Domain Gains

**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.

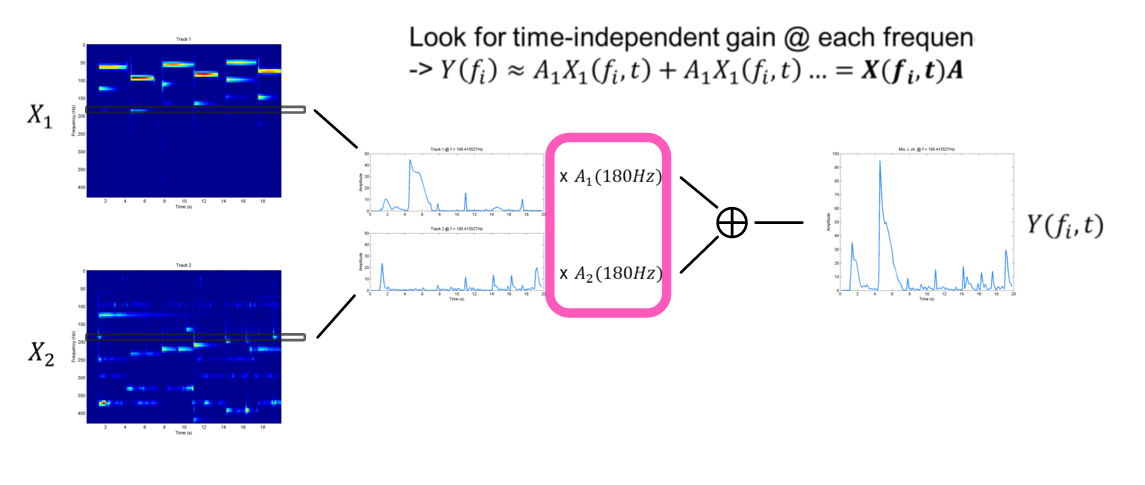


Fig 2. Learning Frequency Domain Gains

Combining these two steps allows us to complete the learning portion of the algorithm. A high-level block diagram of this method is shown below in figure 3.

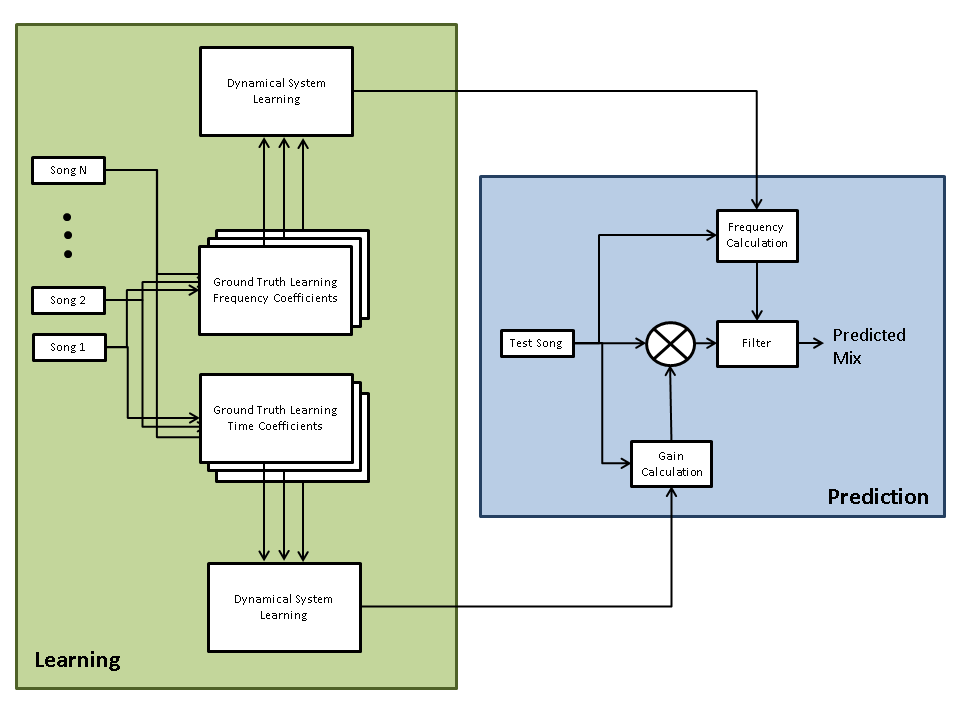


Figure 3. Proposed Learning Algorithm

**3.3 Hyper-parameters**

These following hyper-parameters are relevant to find coefficient.

* RMS window size and Regression Window size

We took Root Mean Square of the signals within certain range, using RMS window as one of our features. Locally weighted linear regression is applied on this feature within a small time range. The larger the window we choose for linear regression, the better time resolution of our results, but the less robustness due to the larger variance in the 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 these zero-signal periods, the coefficients obtained from closed form solutions will be extremely large and unreasonable. To solve this issue, we chose to adjust the gain to zero manually during these periods. In our implementation, we chose to calculate the energy inside each window of time and set an energy threshold for each window. This allows us to verify whether or not there is signal in each individual window. As the threshold becomes increases, more signals would be ignored and thus less accurate gain coefficients. As the threshold decreases, more of these zero-signal periods will be allowed, which will also affect our accuracy.

* FFT Size

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

* Frequency Integration Width

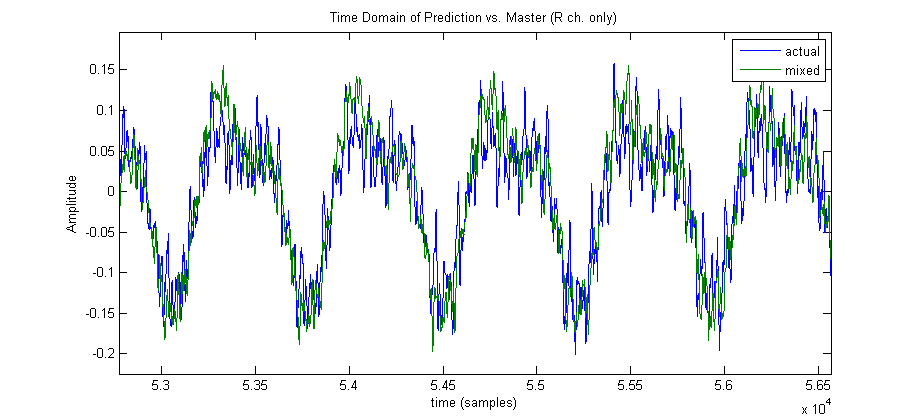
This is another fixed value, which mimics standard filtering capabilities.

**3.4 Proposed Evaluation Method**

We evaluate the reliability of the mixing coefficients we found in two aspects. Our first method would be to use the MSE to determine whether the mixing coefficients we found are similar to the real mixing coefficients. Since in real cases, no mixing coefficients will be given, we will need to manually generate various reasonable mixing coefficients and apply it to our input tracks. These mixed versions can then be used 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 can 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:**



*Figure 5. Comparison of mix using predicted gains values (mixed) and original mix (actual)*

As we have no evaluation method deemed for anything but our final mixes, we subject our other results to the eye test. This allows us to check our progress as we move through the algorithm to ensure that each individual piece of the process is completed successfully and working as intended. We are also able to tune fixed hyper-parameters such as FFT size to obtain the best results.

**4.2 MSE Evaluation**

In general, real mixing coefficients should behave like a wave that fluctuates between zero and one. As mentioned previously, random real mixing coefficients and corresponding outputs need to be generated in order to test the reliability of our model. We generated random mixing coefficients by defining a function against time range. For any given track, the mixing coefficients at time *ti* is defined as follow:

where *k1 ,k2 ,k3 ,k4* are random Gaussian number and *use exponent, use log* are randomly generated indicator variables. Each set of mixing coefficients is also scaled to between [0,1] to make it similar to real mixing process.

We evaluate our model of estimating good mixing coefficients by generating 100 random sets of validation data. MSE are used to show whether the estimated mixing coefficients are closed to real ones. To better illustrate this, we also compute the MSE of constant mixing coefficients, which represents a simple combination tracks without any mixing), as well as the MSE of random mixing coefficients.

The result shown that our model has a robust performance over randomly generated mixing coefficients. The MSE value remains relatively low with small variance within models.

|  |  |  |
| --- | --- | --- |
| Model | Constant | Random |
| 0.042124 | 1.2157 | 0.53913 |



|  |
| --- |
| (Randomly selected from 100 set of mixing coefficients) |
|  |
|  |
|  |
| **4.3 Correlation Evaluation**   |  |  |  | | --- | --- | --- | | Model | Unmixed | Random | | 0.0562 | -0.0049 | -0.0819 |   The table above shows the scores that were achieved using correlation as our evaluation criteria. The equation for the score is as follows:  The score, as described in section 3.4, is composed of the ratio between the predicted dot product to the actual dot product (y is our predicted mix, x is the actual mix). To prevent the loophole where the predicted output can be amplified to boost the score (resulting in an inherently higher dot product), we compare the RMS of the two. If the RMS of the predicted mix is higher than that of the actual mix, the score is decreased by a quadratic amount.  We took the score for our predicted mix and compared it to the score of the unmixed version and a mix where we set random gains. As shown above, our predicted model scores the highest. This shows that our algorithm is able to some extent learn the actual gain models through training. The unmixed model scores the second highest as it maintains a degree of similarity with the mixed version. The mix with random gains scores the lowest, as it is unable to replicate the actual mix at all. |

**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.