MATLAB Model of the Echoplex EP-4 Tape Delay

Michael Zajner
MUMT 618: Computational Modeling of Musical Acoustic Systems
Music Technology, Schulich School of Music, McGill University
Montréal, QC, CA
michael.zajner@mail.mcgill.ca

ABSTRACT

This work presents a MATLAB-based digital model of the Echoplex tape delay unit. The implementation focuses on replicating key characteristics of the physical Echoplex device, including fluctuating time delay driven by capstan and pinch wheel irregularities, saturation effects, and a feedback-based comb filter. The model employs sinc interpolation for anti-aliasing, a leaky integrator for time delay stabilization, and dynamic modulation of delay times. The paper provides an overview of the implementation methodology, with emphasis on design challenges, debugging strategies, and practical usage demonstrations. Results are evaluated through audio tests and visualizations to verify the accuracy of the model.

1. INTRODUCTION

The Echoplex is a renowned tape delay effect unit introduced in the 1950s, celebrated for its warm and characteristically dynamic sound. Originally developed as a hardware device, the Echoplex utilizes a magnetic tape loop to create echoes, with adjustable delay time and feedback. Its mechanical imperfections, such as tape wear and drift, became defining features that contributed to its rich, textured sound. Widely adopted by musicians and producers, the Echoplex became a staple in shaping the soundscapes of the 1960s and 1970s.

In recent years, the shift towards digital audio workstations and virtual instruments has spurred a demand for accurate digital recreations of classic analog effects. Such models not only preserve the sonic heritage of vintage devices but also offer users the convenience, flexibility, and reliability of modern digital platforms. However, replicating the nuanced behaviors of analog systems in the digital domain remains a challenging task, requiring sophisticated algorithms and a deep understanding of the physical processes being emulated.

This paper documents the implementation of a MATLAB-based digital model of the Echoplex tape delay, inspired by prior research and hardware analysis by Arnardottir et al. [1]. The project focuses on recreating key features of the original unit, including delay modulation due to capstan and pinch wheel drift, non-linearities from tape saturation, and anti-aliasing techniques using sinc interpolation. By combining these elements, the model seeks to replicate the warm, unpredictable sound of the original device while providing an educational platform for understanding and experimenting with virtual analog modeling.

The following sections provide an overview of the system's characteristics, a detailed description of the implementation, and an evaluation of the model's performance through audio examples and visualizations. The challenges encountered and their solutions are discussed, followed by suggestions for future improvements to the model.

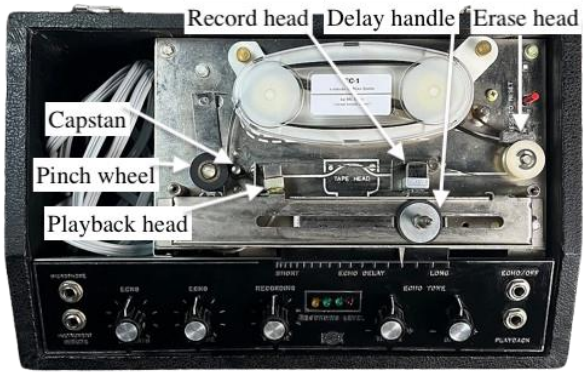


Fig. 1 Maestro Echoplex EP4 Solid State Tape Delay

2. ANALAYSIS

There are two fundamental mechanisms of tape delay: speed-type and length-type. *Speed-type* delays, as exemplified by devices like the Roland Space Echo, alter the delay time by changing the

speed of the tape [2]. This mechanism inherently couples delay time changes with pitch shifts due to the Doppler effect. In contrast, the Echoplex employs a length-type mechanism, where delay time adjustments occur through the physical displacement of the record head. This distinction allows for independent modulation of delay time and pitch, but not without introducing its own unique effects. This distinction allows for independent modulation of delay time and pitch, while also introducing unique artifacts and sonic effects.

The defining feature of the Echoplex is its ability to continuously adjust delay time through the manipulation of the delay handle, which moves the record head relative to the playback head. As highlighted in the work of Arnardottir et al., abrupt movements of the delay handle can result in a range of distinctive sonic effects. For example, quickly moving the delay handle away from the playback head can shift the tape bias signal into the audio band, creating unexpected tonal artifacts. Conversely, when the delay handle is moved towards the playback head at a speed exceeding the tape's motion, the system experiences a "sonic boom" effect. This occurs because the bandwidth of the signal written to the tape becomes unbounded at the point where the record head and the tape move at the same speed, leading to a discontinuity in the output signal.

A further central characteristic of the Echoplex is its feedback mechanism, controlled by the "Repeats" knob. This feedback loop reintroduces delayed signals into the playback chain, enabling a range of effects from subtle decaying echoes to dense, cascading layers of sound. The Repeats knob has a range of 0 to 2, meaning that saturation in the tape and circuitry limit the signal amplitude when the feedback is greater than 1. This can push the system into self-oscillation, producing intense and evolving textures that musicians often exploit for creative purposes. This interplay between delay and feedback forms the foundation of the Echoplex's versatility.

Another defining trait is the nonlinearity introduced by the analog tape medium. Tape saturation, particularly at higher signal levels, introduces harmonic distortion, imparting warmth and character to the sound. Over time, tape wear and mechanical imperfections, such as those from the capstan and pinch wheel, contribute to subtle modulations in tape speed. These irregularities, often referred to as "drift," result in natural chorusing effects and fluctuating time delays, as discussed in detail by Arnardottir et al. These effects enhance the spatial and temporal complexity of the echoes, creating a sense of organic movement.

Mechanical imperfections also lead to comb filtering effects, resulting from the interaction between the feedback loop and tape properties. This interaction introduces tonal coloration, with specific frequencies emphasized based on the delay time and tape characteristics. These resonances, particularly when coupled with feedback, add a unique timbral quality to the delayed signal.

While analog imperfections were integral to the Echoplex's character, digital models must address challenges like anti-aliasing. Digital implementations must smooth abrupt delay-time changes and prevent high-frequency aliasing artifacts to preserve the natural and musical behavior of the original device. Arnardottir et al. emphasize the importance of incorporating anti-aliasing measures, such as sinc interpolation, to ensure fidelity in digital recreations.

The characteristics described above highlight the complex interplay of mechanical, electrical, and acoustic factors that define the Echoplex's sound. As outlined in the work by Arnardottir et al., these characteristics not only inform the sonic signature of the Echoplex but also present unique challenges for digital modeling. The following section discusses how these characteristics are translated into the MATLAB-based digital model, leveraging modern signal processing techniques to emulate the device's analog behavior.

3. OVERVIEW

The signal flow architecture of the MATLAB Echoplex model, as illustrated in the accompanying block diagrams (Figure 2), effectively emulates the fundamental processes of the original hardware. This architecture integrates multiple interconnected modules to replicate the core functionalities of the tape delay system, from the initial input signal processing to the final output. Each module plays a specific role in re-creating the analog characteristics of the Echoplex, including dynamic modulation, drift generation, feedback, and saturation effects.

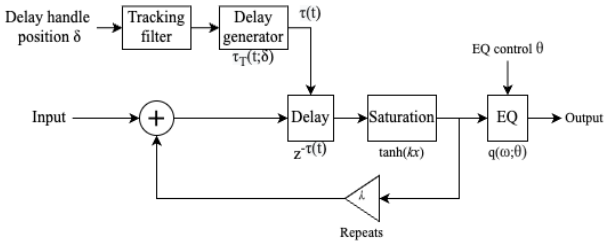


Fig. 2 Echoplex signal flow

The architecture begins with the input signal and delay time modulation stage. The delay handle position (δ) determines the nominal delay time, which is processed via a tracking filter to ensure smooth transitions. Dynamic delay values ($\tau(t)$) are then generated by combining inputs from the tracking filter with drift components and irregularities caused by capstan and pinch wheel fluctuations. These elements simulate the mechanical imperfections that are intrinsic to the original tape system.

The drift and fluctuation generation module introduces both low-frequency drift and periodic modulations to emulate mechanical imperfections. Drift components include white Gaussian noise, filtered by a low-pass filter to remove high-frequency artifacts, and sinusoidal signals that replicate capstan and pinch wheel irregularities at characteristic frequencies such as 2.5 Hz, 5 Hz, and 26 Hz. This module ensures the natural variability in delay time.

In the dynamic delay and feedback stage, the delay module employs a circular buffer to simulate tape motion and accommodate variable delay times. The feedback mechanism, controlled by the "Repeats" knob (λ), reintegrates the delayed signal back into the delay module. Before re-entry, the feedback signal is passed through the saturation stage, adding harmonic richness to the repetitions.

The saturation and equalization module applies nonlinear saturation using a hyperbolic tangent function ($\tanh(kx)$), a cheap approximation of the harmonic distortion characteristic of analog tape systems. Following saturation, an equalization (EQ) stage shapes the tonal output, allowing for further user control over the sound. Together, these processes enhance the authenticity of the model by replicating the warmth and coloration of tape-based systems.

3. METHODOLOGY

This work implements a MATLAB-based digital model of the Echoplex EP-4 tape delay. The methodology involves modeling the key physical and acoustic characteristics of the original device. The design follows prior research on the Echoplex, particularly the work of Arnardottir et al., leveraging signal processing techniques such as sinc interpolation, low-pass filtering, and dynamic feedback control.

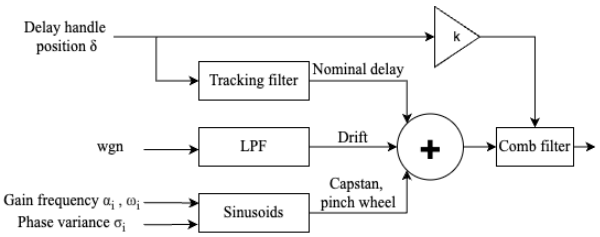


Fig. 3 Delay generation signal flow

3.1 Capstan and Pinch Wheel Drift

A defining feature of the Echoplex's sound is the subtle, time-varying fluctuations in delay caused by mechanical imperfections in the tape transport system (Figure 3). These irregularities, particularly those caused by the capstan and pinch wheel, are modeled as sinusoidal modulations added to the nominal delay. The total drift signal can be expressed as:

$$\begin{aligned}\text{Drift}(t) = A_1 \sin(2\pi f_1 t + \phi_1) \\ + A_2 \sin(2\pi f_2 t + \phi_2) \\ + A_3 \sin(2\pi f_3 t + \phi_3)\end{aligned}$$

where f_1 = 26Hz and f_2 = 2.5Hz, corresponding to irregularities introduced by the pinch wheel and f_3 = 5Hz resulting from the capstan. A_i and ϕ_i are the amplitudes and phases of the sinusoids, adjusted to match the observed characteristics of the Echoplex.[1] To replicate the slight variability observed in the hardware, a low-pass filtered white noise is added to the sinusoidal drift to introduce stochastic variations, and the noise signal is filtered using a

[1] Arandottir et al. unfortunately do not provide detailed information on these values. The values used are therefore approximations to the graphs provided.

Butterworth low-pass filter with a cutoff frequency of 100 Hz, ensuring smooth, natural fluctuation (Figure 4).

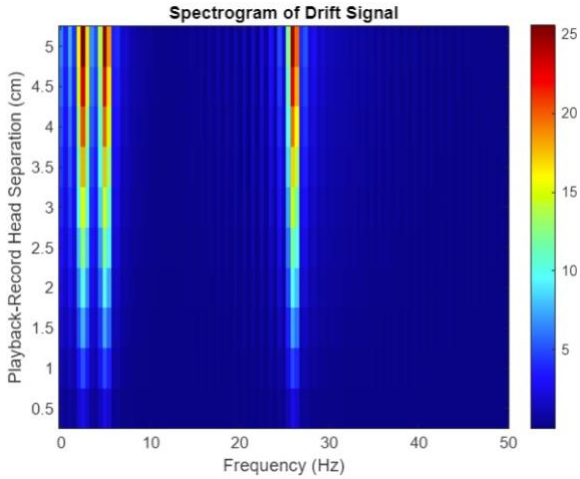


Fig. 4 Spectrogram of Drift Signal Across Playback-Record Head Separations

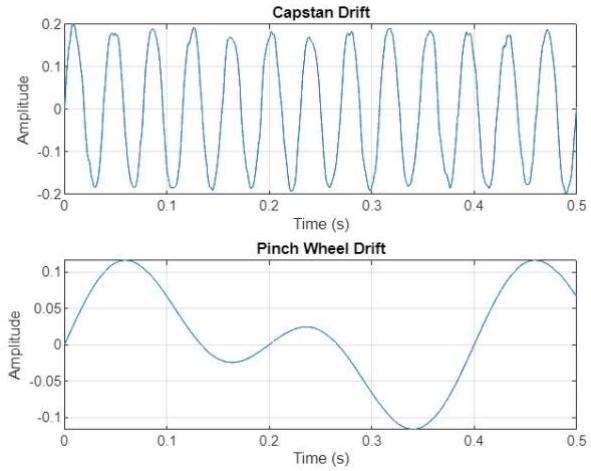


Fig. 5 Time-Domain Representation of Capstan and Pinch Wheel Drift

3.2 Comb Filter Implementation

Arnardottir et al. explain that spectral nulls occur at frequencies proportional to odd integer multiples of the inverse record head-playback head distance. These nulls are likely caused by mechanical disturbances propagating along the tape between the record and playback heads. This behavior is characteristic of a feedback comb filter, implemented here as a simple feedback system dynamically modulated by the delay handle position.

The comb filter delay time (τ_c) is determined by the distance between the record and playback heads, controlled by the delay handle position (δ). This relationship ensures that the tonal characteristics of the feedback comb filter adapt dynamically as the delay handle is moved. The delay time, therefore, is given by:

$$\tau c = f(\delta)$$

where $f(\delta)$ maps the delay handle position to the corresponding delay time. As the delay handle moves closer to the playback head, the delay time decreases, causing the spectral nulls to spread further apart. Conversely, moving the handle away increases the delay time and concentrates the nulls at lower frequencies.

The comb filter is implemented using a circular buffer to store delayed samples. The output at any time step n is calculated as:

$$y[n]=x[n]+g_c \cdot y[n-\tau_c]$$

where g_c is the feedback gain, controlling the intensity of the tonal resonances and τ_c is the delay time, modulated by the delay handle position.

The comb filter introduces periodic resonances and nulls in the frequency spectrum, with null frequencies determined by:

$$f_{null} = \frac{k}{\tau_c}, k = 1,3,5,\ldots$$

As the delay handle adjusts the delay time, the null frequencies shift dynamically, creating the characteristic tonal movement of the Echoplex. This can be seen by applying white noise as the input signal as shown in Figure 6:

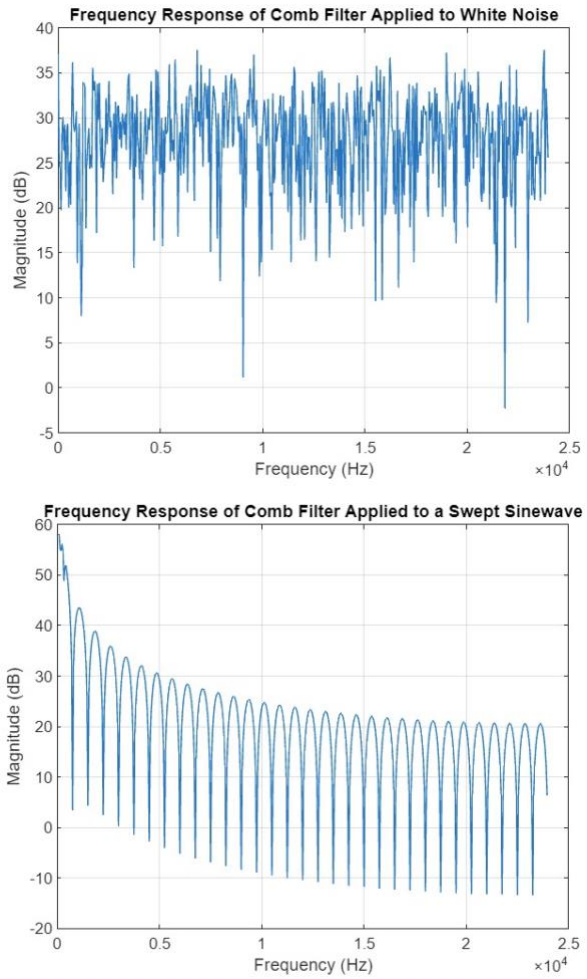


Fig. 6 Comb Filter Frequency Response Applied to White Noise (above) and Sinusoidal Sweep (bottom).

3.3 Feedback, Leaky integration, and Saturation

The feedback (Repeats) mechanism is a fundamental component of the Echoplex tape delay model, as it controls the number of echoes, their intensity, and the overall decay characteristics. This feature allows the system to reintroduce delayed signals into the delay line buffer, enabling cascading repetitions that define the sonic signature of the Echoplex. By shaping the feedback signal through tonal processing, such as the comb filter, and applying saturation effects, the model closely emulates the behavior of the original hardware.

The feedback implementation in the MATLAB model is governed by three core processes: feedback gain scaling, leaky integration for natural decay, and tonal shaping through the comb filter. Each of these components plays a critical role in achieving a dynamic and musically engaging delay response.

The leaky integrator is a crucial component in modeling the natural decay behavior of the Echoplex tape delay. Based on Arnardottir et al., the leaky integrator ensures that the delay smoothly transitions towards a target delay value, emulating the gradual decay and dynamic stability observed in analog tape delays.

The equation governing the leaky integrator can be expressed as:

$$\tau[n+1] = (1-\lambda)\tau T + \lambda\tau[n],$$

This formulation ensures that the delay gradually relaxes towards the target value, with λ dictating the rate of decay. The leaky integrator smooths the feedback signal, creating a more organic and natural decay characteristic of analog systems.

In addition to feedback gain scaling and leaky integration, the application of saturation effects further enhances the realism of the Echoplex tape delay model. Saturation introduces harmonic distortion, emulating the non-linear behavior of analog tape systems. This effect not only enriches the tonal quality of the repeated echoes but also prevents signal overload by soft-clipping peaks in the feedback loop.[2]

The saturation effect in the MATLAB model is implemented as a hyperbolic tangent function. mathematically represented as:

$$y[n] = a \cdot tanh(\frac{y[n]}{a})$$

This mathematical model ensures that high-amplitude signals are compressed non-linearly, introducing subtle harmonic distortion while preserving signal stability. By incorporating this function, the MATLAB implementation replicates the soft-clipping behavior of analog systems, enhancing the perceived warmth and richness of the delay.

Integrating the saturation stage with the leaky integrator creates a feedback loop with dynamic

[2] Unfortunately, Arnardottir provides little to no information on tape saturation here. Other papers discuss this in detail, though their work is restricted to other tape models [3].

tonal shaping. The interaction between these components not only prevents runaway feedback but also introduces the signature harmonic coloration associated with the Echoplex tape delay.

3.4 Delay Line Interpolation

The delay line in the Echoplex model ensures smooth and time-varying delay times by implementing interpolated signal values between discrete samples. This method is fundamental to achieving the continuous-like behavior of analog tape delays within a digital framework. Arnardottir et al. directly reference the resampling method by sinc interpolation [4], which enables non-uniform and time-varying sampling through interpolated lookup techniques.

The delay line interpolation builds on the concept of bandlimited interpolation, which reconstructs a continuous-time signal $x(t)$ from its discrete samples' $x[n]$. Accordingly, the key equation for reconstruction is:

$$x(t) = n\sum x[n] \cdot sinc(\pi Fs \cdot (t - nTs))$$

To achieve fractional delays, this equation requires evaluation of $x(t)$ at arbitrary, non-integer time indices $t = kT_s + \Delta t$, where Δt is a fractional offset. The implementation proposes using precomputed filter coefficients stored in a lookup table, combined with linear interpolation to approximate intermediate values. The MATLAB script models this delay line interpolation by using a circular buffer and a sinc-based interpolation kernel, as seen in Figure 7:

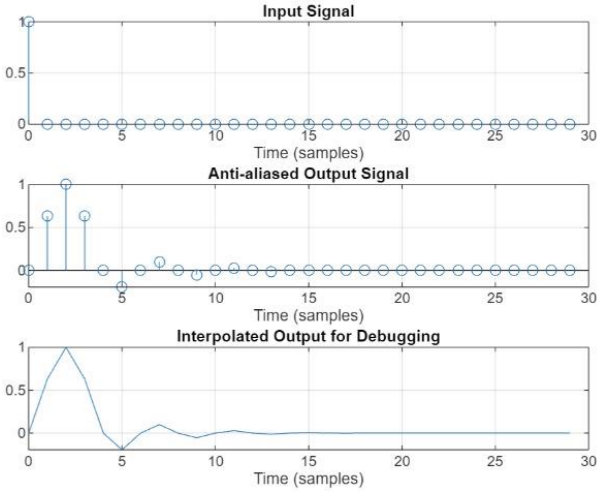


Fig. 7 Impulse Response with Sinc Interpolation

3.5 Anti-Aliasing and High Frequency Behaviour

Anti-aliasing is a critical component of digital delay systems, particularly in tape delay models like the Echoplex, which exhibit time-varying delay lines. Time-varying delays, such as those modulated by capstan drift or pinch wheel irregularities, introduce high-frequency artifacts that can alias into the audible range during digital processing. Arnardottir et al. emphasize the need to carefully address these artifacts to ensure that the model accurately emulates the smooth, warm sound of the original hardware.

In their paper, Arnardottir et al. recommend using interpolation-based anti-aliasing techniques to suppress unwanted high-frequency components. Specifically, they propose linear or higher-order interpolation methods to reconstruct the signal with minimal distortion. They also discuss low-pass filtering as a complementary method to attenuate the high frequencies that can arise from abrupt changes in the delay line's phase.

The key mathematical considerations for anti-aliasing in a time-varying delay are:

1. Filtering the input signal before writing it to the delay buffer (pre-anti-aliasing).
2. Interpolating between delay line samples to avoid abrupt discontinuities (interpolated read or write).
3. Applying a post-low-pass filter to the reconstructed signal for additional high-frequency smoothing.

The MATLAB script attempts to implement anti-aliasing and high-frequency behavior control in alignment with the theoretical recommendations outlined by Arnardottir et al. The model incorporates a 64th-order FIR low-pass filter with a cutoff frequency of 20 kHz. Additionally, sinc interpolation is employed for delay line reads, ensuring smooth transitions between samples and minimizing artifacts caused by time-varying delay modulation. To further enhance the interpolation process, the sinc kernel is windowed using a Hann function, reducing ringing artifacts and improving the naturalness of the reconstructed signal. Together, these features replicate the smooth and warm sound of the Echoplex while mitigating aliasing artifacts inherent to digital systems.

Despite this, the implementation leaves room for refinement. Adjusting the FIR filter's cutoff frequency dynamically based on the rate of delay modulation could not be achieved without extreme

CPU cost. Exploring alternative interpolation methods, such as polynomial or spline interpolation, may offer computational trade-offs that are more suitable for real-time applications. Overall, the anti-aliasing and interpolation techniques adopted in the MATLAB implementation come close to addressing the high-frequency challenges and align with the principles described by Arnardottir et al., while offering a foundation for further optimization and validation.

4. CONCLUSION

The MATLAB implementation of the Echoplex tape delay model provides a flexible and robust platform for replicating the dynamic and tonal qualities of the original hardware. By integrating key features such as dynamic delay modulation, leaky integration, and anti-aliasing, the script adheres closely to the theoretical principles outlined by Arnardottir et al. and offers a comprehensive digital emulation of the Echoplex's analog behavior.

Included in the script is parameter adjustability. Users can fine-tune critical aspects of the delay model, including the delay time and feedback gain, which directly influence the number of echoes and their intensity. Parameters such as the leak coefficient and modulation amount further allow for precise control over natural decay and dynamic instability, emulating the subtle irregularities of analog drift. Additionally, tonal shaping through the comb filter and the inclusion of high-frequency behavior controls, such as FIR-based anti-aliasing, ensure a nuanced reproduction of the Echoplex's unique sound.

The script also incorporates a variety of input signal options to facilitate debugging and evaluation of its components. Impulse signals are particularly useful for observing the decay characteristics and temporal spacing of echoes, while white noise enables a detailed analysis of spectral response, including the interplay between feedback and tonal shaping mechanisms. Sine waves can be used to identify resonance effects and validate the frequency-dependent behavior of the delay line, and sine sweeps are instrumental in assessing the performance of anti-aliasing and time-varying delay modulation. This testing flexibility enables a methodical approach to refining and optimizing the model.

Future work should focus on systematically validating the MATLAB implementation against physical measurements of an Echoplex unit. Spectral analysis could confirm that the model's frequency response aligns with the hardware's analog characteristics, while time-domain analyses would ensure accurate decay and modulation behavior. Stress tests using extreme parameter values could further assess the robustness of anti-aliasing, interpolation, and drift components under challenging conditions. Subjective listening tests involving expert feedback could fine-tune the model's sonic output to align with user expectations. Together, these methodologies would not only validate the existing implementation but also provide a pathway for enhancing its fidelity, ensuring that the digital emulation captures the complexity and nuance of the original Echoplex tape delay.

REFERENCES

[1] Arnardottir, S., Abel, J. S., & Smith, J. O. (2008). *A Digital Model of the Echoplex Tape Delay*. Audio Engineering Society Convention 125. Retrieved from https://secure.aes.org/forum/pubs/conventions/?elib=14800.

[2] Downing, Jon, & Terjesen, Christian (2016). "Real-Time Digital Modeling of the Roland Space Echo." *University of Rochester ECE 472 - Audio Signal Processing*.

[3] Chowdhury, J. (2019). *Real-time physical modelling for analog tape machines*. Proceedings of the 22nd International Conference on Digital Audio Effects (DAFx-19), Birmingham, UK, September 2–6, 2019. Available at: https://ccrma.stanford.edu/~jatin/420/tape/

[4] Smith, J. O. (1984). *A Flexible Sampling-Rate Conversion Method*. Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP). Retrieved from https://ccrma.stanford.edu/~jos/resample/

Echoplex Tape Delay Model

```
% This code implements an Echoplex tape delay model based on the work of Steinunn
Arnardottir,
% Jonathan S. Abel, & Julius O. Smith,  “A digital model of the echoplex tape
delay,” in
% Audio Engineering Society Convention 125, Oct 2008.
% https://secure.aes.org/forum/pubs/conventions/?elib=14800

% For more information and the downloadable plugin, please visit
% https://michaelzajner.com/research/echoplex/

% The model includes interpolated write, anti-aliasing filtering, leaky integrator,
% and additional irregularities such as capstan drift and pinch wheel drift.
```

Section 1: Initialization and Parameters Setup

```
% Adjustable Parameters
% Many parameters are approximations based on the materials provided in the
% article.
% clear all;

sampleRate = 48000; % Sampling rate in Hz
initialDelayMs = 300; % Initial delay in milliseconds (can be adjusted to control
delay time)
combFeedbackGain = 0.1; % Comb filter feedback gain (adjustable to control comb
filtering effect)
delayFeedbackGain = 0.95; % Feedback/repeats control (adjustable to control repeats
amount)
leakCoefficient = 0.95; % Coefficient for leaky integrator (adjustable for natural
decay effect)
dryWetMix = 0.75; % Dry/Wet Mix Ratio (0 = All Dry, 1 = All Wet)
saturationAmount = 0.5; % Amount of saturation applied (0 = No saturation, 1 = Full
saturation)
modulationAmount = 0.1; % Modulation control of capstan & pinch wheel (0 = No
modulation, 1 = Full modulation)

% Simulate moving delay time
delaySweepRangeMs = 100; % Maximum variation for sweeping delay in milliseconds
sweepFrequency = 10; % Frequency of the sweeping delay in Hz

% Convert delay time to samples
initialDelaySamples = round(initialDelayMs * (sampleRate / 1000)); % Convert delay
to samples

% Create UI controls for parameters
%{
fig = uifigure('Name', 'Echoplex Tape Delay Controls');
```

```
initialDelaySlider = uislider(fig, 'Position', [100, 300, 300, 3], 'Limits', [50,
800], 'Value', initialDelayMs);
initialDelaySlider.ValueChangedFcn = @(src, event) assignin('base',
'initialDelayMs', src.Value);
initialDelaySlider.Tooltip = 'Delay (ms)';

delayFeedbackSlider = uislider(fig, 'Position', [100, 250, 300, 3], 'Limits', [0,
1.5], 'Value', delayFeedbackGain);
delayFeedbackSlider.ValueChangedFcn = @(src, event) assignin('base',
'delayFeedbackGain', src.Value);
delayFeedbackSlider.Tooltip = 'Repeats';

dryWetMixSlider = uislider(fig, 'Position', [100, 200, 300, 3], 'Limits', [0, 1],
'Value', dryWetMix);
dryWetMixSlider.ValueChangedFcn = @(src, event) assignin('base', 'Mix', src.Value);
dryWetMixSlider.Tooltip = 'Dry/Wet Mix Ratio';

modulationAmountSlider = uislider(fig, 'Position', [100, 150, 300, 3], 'Limits',
[0, 0.05], 'Value', modulationAmount);
modulationAmountSlider.ValueChangedFcn = @(src, event) assignin('base',
'modulationAmount', src.Value);
modulationAmountSlider.Tooltip = 'Modulation Amount';
%}
```

Section 2: Load and Prepare Audio File

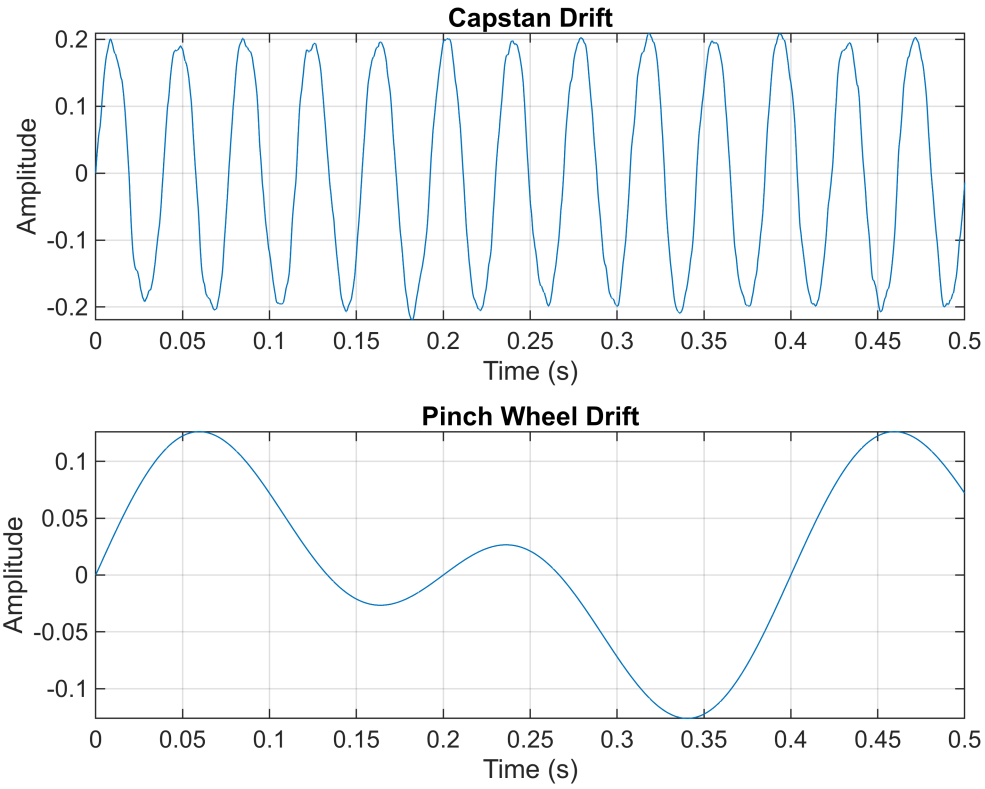
```
% For testing: Load an audio file (mono channel), and set up parameters.
% [inputSignal, sampleRate] = audioread('Test.wav'); % Replace with your audio file
% inputSignal = inputSignal(:, 1)'; % Use the first channel if stereo, transpose to
a row vector

% For testing: Generate white noise
% testDuration = 5; % Duration of the test signal in seconds
% whiteNoiseInput = randn(1, sampleRate * testDuration); % Generate white noise

% For testing: Generate an impulse response
% testDuration = 1; % Duration of the test signal in samples
% impulseInput = zeros(1, sampleRate);
% impulseInput(1) = 1; % Set the first sample to 1 to create an impulse

% For testing: % Generate sine sweep
% sineSweepInput = chirp(timeVector, 20, timeVector(end), 20000); % Frequency sweep
from 20 Hz to 20 kHz

% For testing: % Generate sinewave
% sineFreq = 100; % Frequency in Hz
% sineWaveInput = sin(2 * pi * sineFreq * (0:1/sampleRate:1)); % 1 second sine wave

% Replace the 'inputSignal' with the impulse or white noise for testing
% inputSignal = whiteNoiseInput;
```

```
% inputSignal = impulseInput;
% inputSignal = sineSweepInput;
% inputSignal = sineWaveInput;

% Time vector based on audio length
timeVector = (0:length(inputSignal)-1) / sampleRate;

%% Section 2.1: Sweeping Delay Modulation Setup
% Generate sweeping modulation for delay time
sweepModulation = sawtooth(2 * pi * sweepFrequency * timeVector, 0.5) *
(delaySweepRangeMs / 2);

% Create a sharp drop lasting 30 milliseconds
sharpDropDurationSamples = round(30 * (sampleRate / 1000));
sharpDrop = linspace(1, 0, sharpDropDurationSamples);
sweepModulation = [sweepModulation(1:end-sharpDropDurationSamples), sharpDrop];

sweepingDelaySamples = initialDelaySamples + round((sweepModulation * (sampleRate /
1000)));
```

Section 3: Capstan Pinch Wheel, and Noise Drift Generation

```
% Generate various drift components affecting the tape delay system.

% Capstan Drift (Sinusoidal)
capstanFrequency = 26; % Frequency for capstan drift in Hz
capstanAmplitudeSec = 0.00005; % Amplitude of the capstan drift
capstanPulse = sin(2 * pi * capstanFrequency * timeVector); % Generate sinusoidal
signal
phaseNoiseCutoff = 100; % Cutoff frequency for phase noise filter in Hz
[b, a] = butter(2, phaseNoiseCutoff / (sampleRate / 2)); % Butterworth IIR filter
(2nd order)
whiteNoise = randn(1, length(timeVector)); % Generate white noise
filteredPhaseNoise = filter(b, a, whiteNoise); % Filter the noise to add drift

combinedPhase = capstanPulse + filteredPhaseNoise; % Combined phase signal
capstanDrift = tanh(sin(combinedPhase) * capstanAmplitudeSec * sampleRate *
modulationAmount); % Capstan drift calculation with limiting

% Pinch Wheel Drift
pinchWheelFreq1 = 2.5; % First pinch wheel irregularity in Hz
pinchWheelFreq2 = 5.0; % Second pinch wheel irregularity in Hz
pinchWheelPulse1 = sin(2 * pi * pinchWheelFreq1 * timeVector);
pinchWheelPulse2 = sin(2 * pi * pinchWheelFreq2 * timeVector);
pinchWheelDrift = tanh((pinchWheelPulse1 + pinchWheelPulse2) * 0.00005 *
initialDelayMs / 1000 * sampleRate * modulationAmount); % Pinch wheel drift
calculation with limiting

% Noise
```

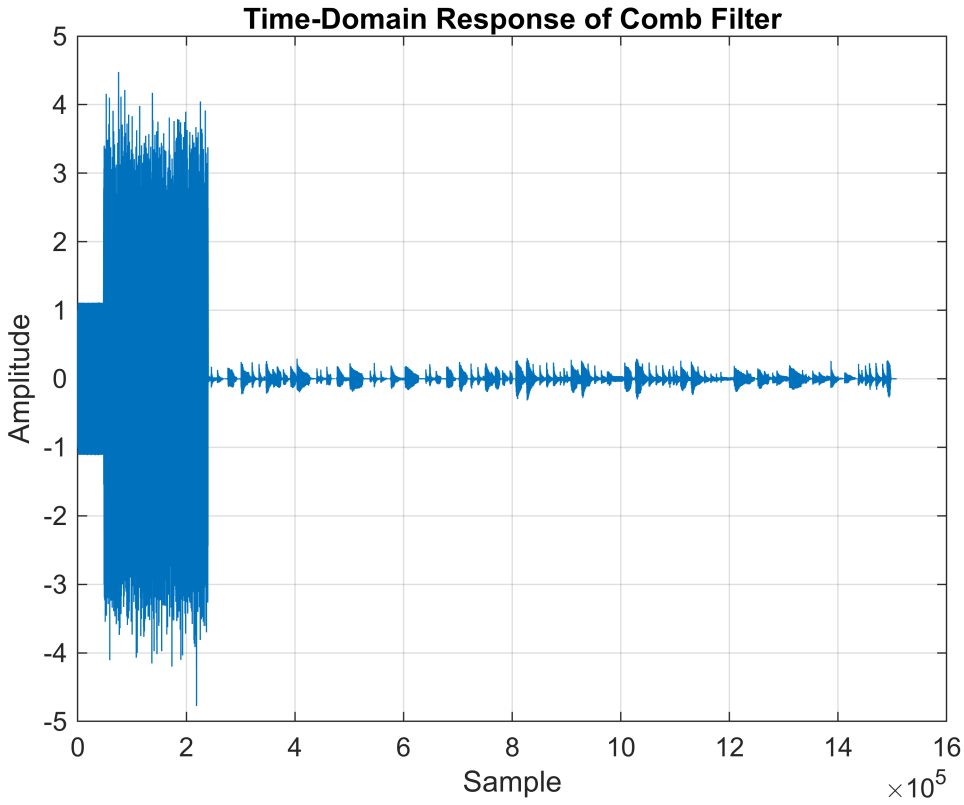
```
% Add noise to simulate mechanical imperfections.
noiseAmplitude = 0.00075 * (initialDelayMs / 1000);
mechanicalNoiseSignal = noiseAmplitude * whiteNoise;
filteredMechanicalNoise = filter(b, a, mechanicalNoiseSignal);

%% Plot Capstan and Pinch Wheel Drift
figure;
subplot(2, 1, 1);
plot(timeVector, capstanDrift);
xlim([0 0.5]); % Focus on the first 0.5 seconds
title('Capstan Drift');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;

subplot(2, 1, 2);
plot(timeVector, pinchWheelDrift);
xlim([0 0.5]); % Focus on the first 0.5 seconds
title('Pinch Wheel Drift');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;
```



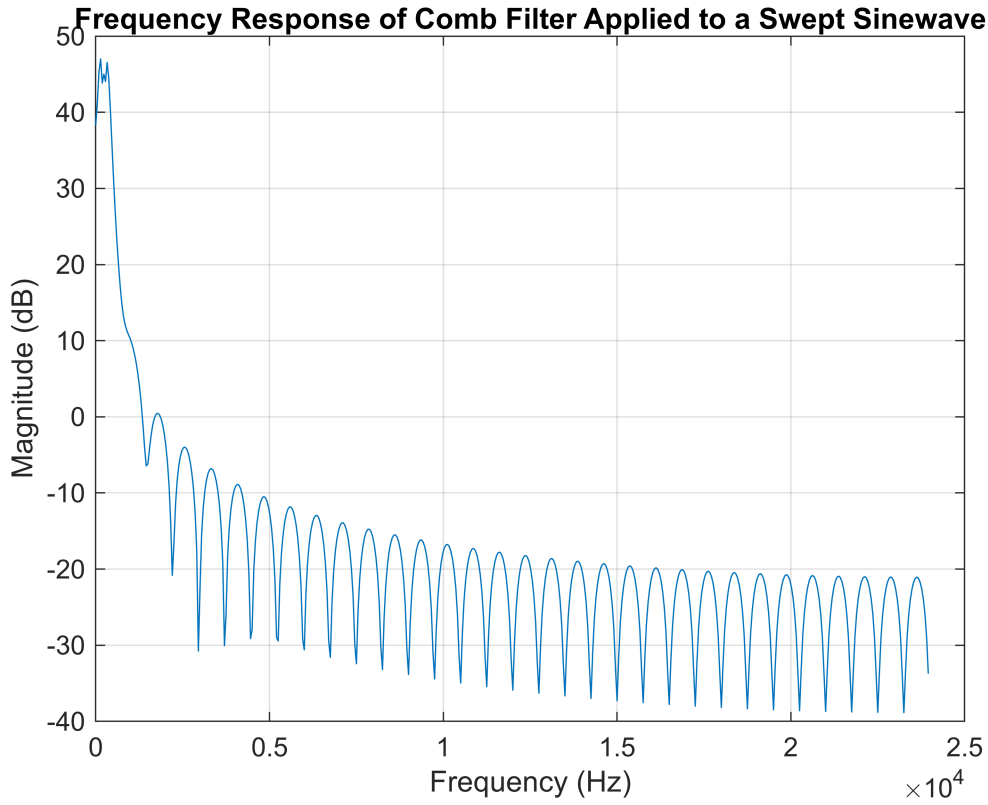
Section 4: Comb Filter

```
% Parameters for Comb Filter
```

```
initialCombDelay = 0.02 * sampleRate; % Base comb delay length (e.g., 20 ms)
combBufferLength = round(initialCombDelay) + 1; % Buffer length for comb filter
combBuffer = zeros(1, combBufferLength); % Initialize comb buffer
combWritePointer = 1; % Write pointer for comb filter

% Update comb filter delay time dynamically
for n = 1:length(inputSignal)
    % Update comb filter delay time based on delay handle position
    currentDelayHandle = sin(2 * pi * sweepFrequency * (n / sampleRate)); % Example
modulation
    combDelaySamples = round(initialCombDelay + modulationAmount *
currentDelayHandle);

    % Ensure delay time is bounded
    combDelaySamples = max(1, min(combDelaySamples, combBufferLength - 1));

    % Retrieve delayed value from comb buffer
    combReadPointer = combWritePointer - combDelaySamples;
    if combReadPointer <= 0
        combReadPointer = combReadPointer + combBufferLength;
    end

    % Calculate comb filter output
    combDelayedValue = combBuffer(combReadPointer);
    combFilterOutput = inputSignal(n) + combFeedbackGain * combDelayedValue;

    % Update comb buffer
    combBuffer(combWritePointer) = combFilterOutput;
    combWritePointer = combWritePointer + 1;
    if combWritePointer > combBufferLength
        combWritePointer = 1; % Wrap around buffer
    end

    % Store the comb filter output for analysis
    combOutputSignal(n) = combFilterOutput;
end

% Plot the time-domain response of the comb filter
figure;
plot(combOutputSignal);
xlabel('Sample');
ylabel('Amplitude');
title('Time-Domain Response of Comb Filter');
grid on;
```




```
% Frequency response of comb filter
fftCombOutput = abs(fft(combOutputSignal, 1024));
freqAxis = (0:511) * (sampleRate / 1024);
figure;
plot(freqAxis, 20*log10(fftCombOutput(1:512)));
xlabel('Frequency (Hz)');
ylabel('Magnitude (dB)');
title('Frequency Response of Comb Filter Applied to a Swept Sinewave');
grid on;
```



Section 5: Anti-Aliasing Filters

```
% Anti-Aliasing Filter Setup
% Implement an FIR filter for anti-aliasing to reduce high-frequency components
antiAliasFilterOrder = 64; % Order of the anti-aliasing filter
antiAliasCutoffFreq = 20000; % Initial cutoff frequency for anti-aliasing filter
(in Hz)
antiAliasCoeffs = fir1(antiAliasFilterOrder, antiAliasCutoffFreq / (sampleRate /
2)); % Design a low-pass FIR filter

% Sinc Interpolation for Anti-Aliasing
% Parameters for sinc interpolation
desiredDelaySamples = 4.7; % Desired delay in samples
numSteps = length(inputSignal); % Number of time steps
delayLineLength = initialDelaySamples + 50;
delayLineBuffer = zeros(1, delayLineLength);
outputSignal = zeros(1, numSteps); % Output signal

% Initialize pointers
writePointer = 1;
readPointer = writePointer - initialDelaySamples;
if readPointer <= 0
    readPointer = readPointer + delayLineLength; % Wrap around for negative values
```

```
end

% Setup FIR filter for anti-aliasing (using sinc interpolation)
sincFilterLength = 32; % Length of sinc filter (must be even)
normalizedCutoffFreq = 0.5; % Normalized cutoff frequency for sinc interpolation
sincKernel = sinc(normalizedCutoffFreq * (-sincFilterLength/2:sincFilterLength/2));
sincKernel = sincKernel .* hann(length(sincKernel))'; % Apply Hann window for
smoother response
```

Section 6: Main Loop for Delay Processing

```
% Incorporate the Comb Filter, Leaky Integrator, and Feedback
for n = 1:length(inputSignal)
    % Step 1: Update Read Pointer with Drift
    drift = modulationAmount * (capstanDrift(n) + pinchWheelDrift(n)) +
filteredMechanicalNoise(n);

    % Simulate a sweep in the delay time (remove if desired)
    delaySamples = max(1, sweepingDelaySamples(n)); % Prevent negative or zero delay
    readPointer = writePointer - delaySamples + drift;

    % Without sweeping delay Time
    readPointer = readPointer + 1 + drift;

    % Ensure read pointer wraps around the delay line buffer
    if readPointer > delayLineLength
        readPointer = readPointer - delayLineLength;
    elseif readPointer <= 0
        readPointer = readPointer + delayLineLength;
    end

    % Step 2: Sinc Interpolation for Anti-Aliasing
    interpolatedOutput = 0;
    for k = -sincFilterLength/2:sincFilterLength/2
        currentIndex = floor(readPointer) + k;

        % Handle circular buffer wrapping for the delay line
        if currentIndex <= 0
            currentIndex = currentIndex + delayLineLength;
        elseif currentIndex > delayLineLength
            currentIndex = currentIndex - delayLineLength;
        end

        % Accumulate the weighted samples using the sinc kernel
        interpolatedOutput = interpolatedOutput + delayLineBuffer(currentIndex) *
sincKernel(k + sincFilterLength/2 + 1);
    end

    % Step 3: Apply Feedback Gain (Repeats Control)
```

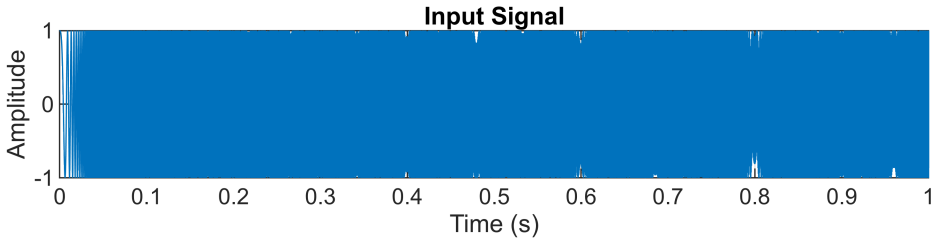
```
    feedbackSignal = delayFeedbackGain * interpolatedOutput;

    % Step 4: Apply Comb Filter in Feedback Loop
    combDelayedValue = combBuffer(combWritePointer); % Get delayed value from comb
buffer
    combFilterOutput = feedbackSignal + combFeedbackGain * combDelayedValue; %
Apply comb filter feedback
    combBuffer(combWritePointer) = combFilterOutput; % Store in comb buffer for
future use

    % Step 5: Apply Leaky Integrator
    if n == 1
        previousFeedback = 0;
    end
    feedbackSignal = leakCoefficient * combFilterOutput + (1 - leakCoefficient) *
previousFeedback;
    previousFeedback = feedbackSignal;
    saturatedFeedback = saturationAmount * tanh(feedbackSignal);

    % Step 6: Combine Dry and Wet Signal (Dry/Wet Mix)
    drySignal = inputSignal(n);
    wetSignal = saturatedFeedback;
    outputSignal(n) = (1 - dryWetMix) * drySignal + dryWetMix * wetSignal;

    % Step 7: Write Feedback and Current Input Back to Delay Line
    delayLineBuffer(writePointer) = saturatedFeedback + inputSignal(n);

    % Update the read and write pointers for the delay line, handle circular
wrapping
    writePointer = writePointer + 1;
    combWritePointer = combWritePointer + 1;

    if writePointer > delayLineLength
        writePointer = 1;
    end
    if combWritePointer > combBufferLength
        combWritePointer = 1; % Wrap around the buffer
    end
end

% Apply Anti-Aliasing Filter to Output Signal
outputSignal = filter(antiAliasCoeffs, 1, outputSignal); % Apply anti-aliasing
filter to output signal
```

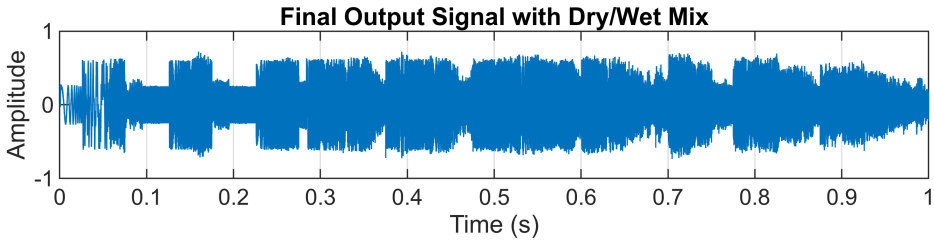
Section 7: Plotting the Results

```
% Plot the input signal for reference
figure;
subplot(3,1,1);
```

```
plot(timeVector, inputSignal);
title('Input Signal');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;

% Plot the final output signal with dry/wet mix applied
subplot(3,1,3);
plot(timeVector, outputSignal);
title('Final Output Signal with Dry/Wet Mix');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;

sgtitle('Echoplex Tape Delay with Sinc Interpolation and Dynamic Anti-Aliasing
Visualization');
```

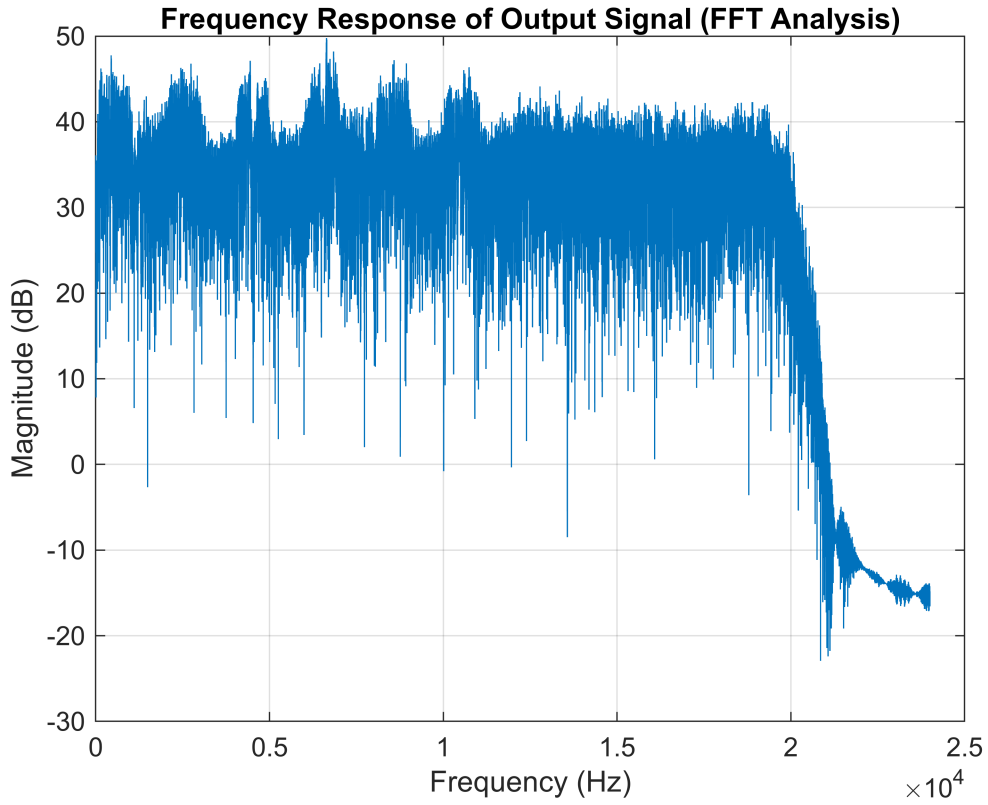
Tape Delay with Sinc Interpolation and Dynamic Anti-Aliasing Vis





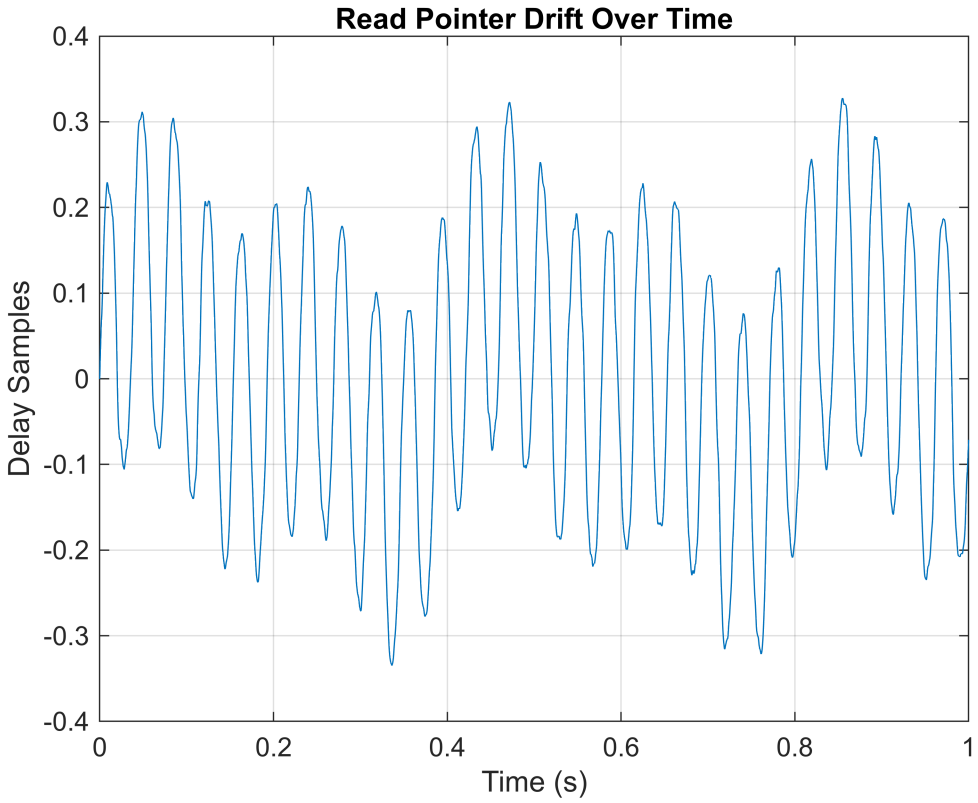

```
% Frequency Response Analysis (FFT)
figure;
outputFFT = abs(fft(outputSignal));
frequencies = linspace(0, sampleRate/2, length(outputFFT)/2 + 1);
plot(frequencies, 20*log10(outputFFT(1:length(frequencies))));
title('Frequency Response of Output Signal (FFT Analysis)');
xlabel('Frequency (Hz)');
ylabel('Magnitude (dB)');
```

```
grid on;
```




```
%{
% Plot input vs. output for the sine sweep
figure;
subplot(2, 1, 1);
plot(timeVector, sineSweepInput);
title('Sine Sweep Input');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;

subplot(2, 1, 2);
plot(timeVector, outputSignal);
title('Sine Sweep Output');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;
%}

%{
% Plotting results for the sinc interpolation with impulse response
timeSteps = 0:numSteps-1;

figure;
subplot(3,1,1)
```

```
stem(timeSteps, inputSignal(1:numSteps))
grid
xlabel('Time (samples)');
ylabel('Amplitude');
title('Input Signal (Impulse)');

subplot(3,1,2)
stem(timeSteps, outputSignal)
grid
xlabel('Time (samples)');
ylabel('Amplitude');
title('Sinc Interpolated Output Signal');

subplot(3,1,3)
plot(timeSteps, outputSignal)
grid
xlabel('Time (samples)');
ylabel('Amplitude');
title('Interpolated Output for Debugging');

sgtitle('Sinc Interpolation Debugging for Impulse Response');
%}

% Plot the read pointer drift over time
figure;
plot(timeVector, capstanDrift + pinchWheelDrift);
title('Read Pointer Drift Over Time');
xlabel('Time (s)');
ylabel('Delay Samples');
grid on;
```



Section 7: Output

```
% Normalize and Clip the Output Signal
% Normalize to prevent clipping during audio writing
maxAmplitude = max(abs(outputSignal)); % Find the maximum absolute value
if maxAmplitude > 1
    outputSignal = outputSignal / maxAmplitude; % Normalize to range -1 to 1
end

% Optional: Clip to ensure no value exceeds -1 to 1
outputSignal = max(min(outputSignal, 1), -1);

%% Save the Output Audio File
outputFilename =
['C:\Users\mikez\Documents\MATLAB\Echoplex\AudioExamples\modulation.wav']; % Change
to desired output location
audiowrite(outputFilename, outputSignal, sampleRate); % Save the output signal
fprintf('Processed test output saved to: %s\n', outputFilename);
```

```
Processed test output saved to: C:\Users\mikez\Documents\MATLAB\Echoplex\AudioExamples\modulation.wav
```