Team 1: 3-D Audio Optimization Tool

Final Report

University of Michigan
EECS 452 Digital Signal Processing Design Laboratory
Arjun Chandrasekhar, Ravynne Jenkins, Stephen Lee, Jonathan Wong

24 April 2021

Table of Contents

Introduction

Background and Motivation

When editing and mastering audio in professional settings, experienced and knowledgeable audio engineers and technicians use information such as the specific frequency response of speakers along with other tools to analyze the frequency response of a room and tune the audio system to their desired settings. For a speaker system used by a general consumer, the details of the speaker system are unknown, and the frequency response of the room is variable due to the obstructions and items placed within it. This leads to a sub-par audio system and sound environment.

The idea for our 3-D Audio Optimization tool came from wanting to figure out an automated, simple, and straightforward method of letting anyone optimize the frequency response of the speakers and correct for minor variations due to the environment. The goal would be to match the frequency response of a speaker's output with that of a studio speaker's output. The method we develop would be applicable to any type of speaker system, given predictable frequency ranges for their audio outputs.

Literature Review

There are a few products that already exist with the aim to optimize or calibrate a consumer's new home theater audio system. Audyssey, THX Tune Up, Multichannel Acoustic Calibration (MCACC), Digital Cinema Auto Calibration (DCAC) and the Yamaha Parametric Room Acoustic Optimizer (YPAO) all accomplish this task in different ways.

The Audyssey eVR system removes the effects of room reverberation to increase voice recognition accuracy and denoises by removing ambient noise from the surroundings. The THX Tune Up is a mobile application that calibrates based on the filters that exist inside of each electronic (Santos, 2020). The app runs sound tests to test and optimize the noise output. The MCACC, DCAC and the YPAO all apply active noise equalization schemes when calibrating the audio system (Guttenberg, 2017). The YPAO analyzes the acoustic parameters of the room and adjusts volume balance and optimizes the speaker settings based on its analysis. The DCAC is equipped with a multi channel A/V receiver to balance sound automatically and perform its acoustic calibration.

A large drawback to the majority of the existing products is that their functionality relies on detecting the connection of a device produced by the same manufacturer. These products then use the audio filtering systems of that device to optimize the acoustics. This allows our product to offer something new to that other products on the market are not offering. This noise optimization tool is compatible with all speakers and builds filters specifically based on the test tone outputs. Our final product is also bringing a different feature with its iterative filter generation approach. Products currently in use on the market have set filters that it applies to optimize and calibrate audio systems. Our system iterates through the generation of filters and compares test tones and sound quality to apply the best possible filter to the system. This allows for a more refined and accurate sound quality.

Project Description

Optimization Phase

Our goal is to create an audio equalizer on a Teensy that is tailored to the user's speaker system and room. We begin with the optimization phase as shown in figure 1. The user connects a Raspberry Pi 4 to the Teensy to generate the filter via USB. The Raspberry Pi 4 is also connected to a microphone that records the speaker's output. The user connects the Teensy audio input to their device that outputs the test tone and connects the Teensy audio output to the speaker system they want to optimize.

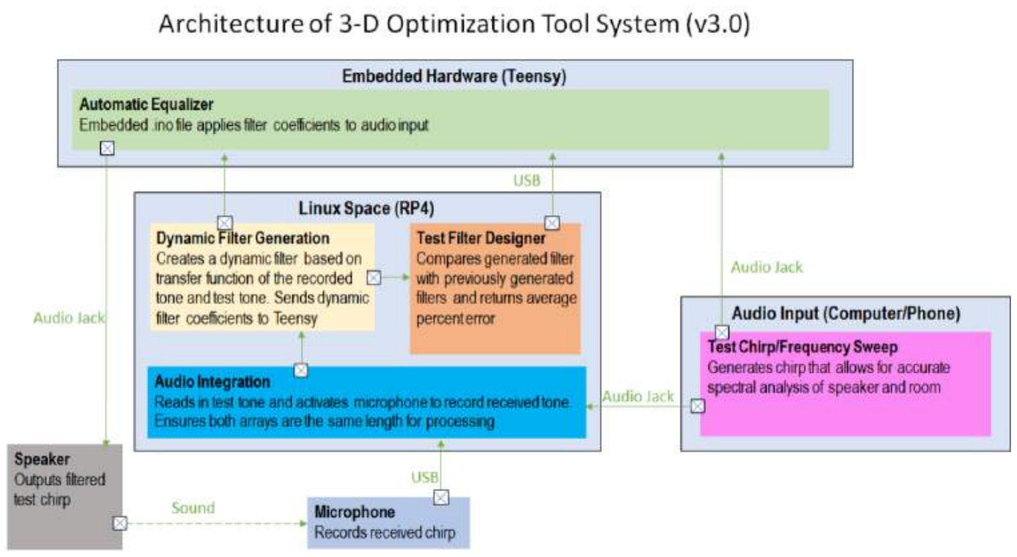

Architecture of 3-D Optimization Tool System (v3.0)
Embedded Hardware (Teensy)
Automatic Equalizer
Embedded .ino file applies filter coefficients to audio input
USB
Linux Space (RP4)
Dynamic Filter Generation
Creates a dynamic filter based on transfer function of the recorded tone and test tone. Sends dynamic filter coefficients to Teensy
Test Filter Designer
Compares generated filter with previously generated filters and returns average percent error
Audio Jack
Audio Jack
Audio Input (Computer/Phone)
Test Chirp/Frequency Sweep
Generates chirp that allows for accurate spectral analysis of speaker and room
Audio Integration
Reads in test tone and activates microphone to record received tone. Ensures both arrays are the same length for processing
Audio Jack
Speaker
Outputs filtered test chirp
USB
Sound
Microphone
Records received chirp

Figure 1: Block diagram of optimization phase

The user then begins the optimization phase and the test tone is played by the speaker. At the same time the test tone is played, the Audio Integration block activates the microphone to begin recording the output from the speaker into a received tone. Both the test tone and received tones are converted from .wav files into usable Numpy arrays.

The Dynamic Filter Generation block uses the Numpy arrays and generates a filter optimized for the user. This is calculated using the equation: $y(t) = x(t) \oplus h(t)$ where $y(t)$is the test tone, $x(t)$is the received tone and $h(t)$is the filter. As convolution is difficult to calculate, the Dynamic Filter Generation uses the frequency domain as shown below:

$$y(t) = x(t) \oplus h(t)$$
$$Y(\omega) = X(\omega) * H(\omega)$$
$$H(\omega) = Y(\omega) \,/\, X(\omega)$$
$$h(t) = \mathcal{L}^{-1}\{Y(\omega) \,/\, X(\omega)\}$$

Our goal is to optimize the speaker so that the speaker outputs exactly the test tone. Since the received tone is the audio actually played by the speaker, we can apply the filter $h(t)$to ensure the output of the speaker is the test tone.

A demo of this Dynamic Filter Generation is shown in figure 2. The blue signal is the test tone $y(t)$and the green signal is the received tone $x(t)$. In this example, the received tone had extra noise in the midband frequencies compared to the test tone. A filter $h(t)$ was generated using the method above. The black signal represents the filtered audio $x(t) \oplus h(t)$. As shown, the filtered audio sufficiently removes the noise and preserves the overall shape of the test tone.

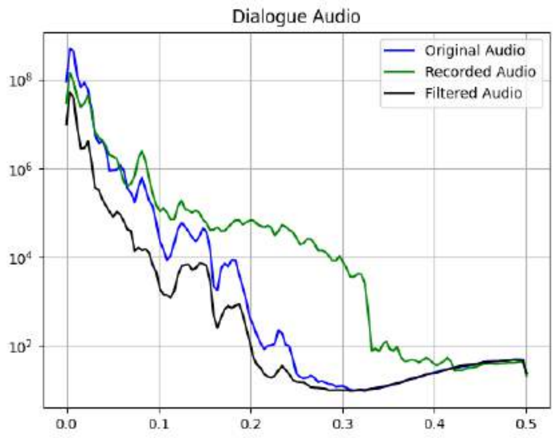


Figure 2: Dynamic Filter Generation model

Deliverable

Since the filter is applied to a one-dimensional audio signal, the resulting filter is a one-dimensional array of coefficients. These coefficients are flashed onto the Teensy so that the filter can be convoluted with any incoming audio signal. Once the coefficients have been flashed, the optimization step is complete and the deliverable is finalized. Figure 3 below shows the deliverable block diagram. The Teensy will apply the filter coefficients to any audio input without the need of a Raspberry Pi 4 or other computer. The user can use any audio input, such as a phone, and play any audio they wish. The Teensy will then apply the filter and output the filtered audio to the speaker. The user will then have optimized audio equalizer in their room.

Currently, the system only performs one iteration of the optimization phase and an iterative process is under development.

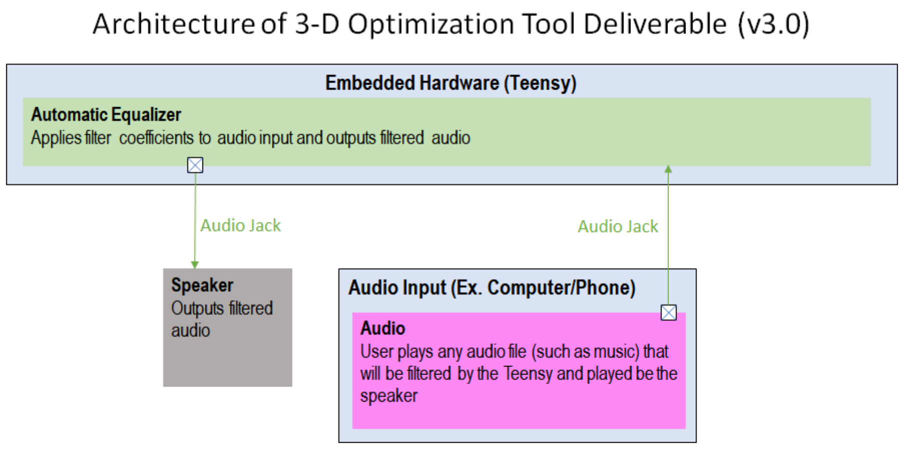

Architecture of 3-D Optimization Tool Deliverable (v3.0)
Embedded Hardware (Teensy)
Automatic Equalizer
Applies filter coefficients to audio input and outputs filtered audio
Audio Jack
Audio Jack
Speaker
Outputs filtered audio
Audio Input (Ex. Computer/Phone)
Audio
User plays any audio file (such as music) that will be filtered by the Teensy and played be the speaker

Figure 3: Deliverable Block Diagram

Test Tones

In order for the calibration of an audio system and a room's frequency response to be performed correctly, the audio signals which are being sent through the speakers need to sweep its entire frequency range. In a 5.1 surround sound speaker system, four speakers operate the left and right channels, one speaker operates human voice frequencies, and one subwoofer operates low frequencies, such as bass.

The filter generation tones had to be designed such that they would fit each of these three types of speakers. We had many options for the type of test signals which were used in the project: White noise, which is a signal of consistent amplitude covering every audible frequency (~ 20 - 20000 Hz), and pink noise, which similarly covers all frequencies but decreases in amplitude as frequency increases, were chosen between to form the basis for all of the developed test tones. In addition, chirps or frequency sweeps were used, which are signals that move through a specified frequency range changing in amplitude along the way. Chirps are useful for specifically testing the known frequency range of a speaker, seeing at each frequency how the speaker's output will be distorted in a room. Forming combinations of these different types of signals proved to be even more effective for our system, as the application of white or pink noise to a chirp can help create a constant reference for how the noise of the room affects the signal.

10-20 ms chirps were developed for filter generation as there is a reduced time delay for the coefficients to be created in comparison to the longer tones, which are better for testing the generated filter in the final deliverable. Additionally, the Teensy has a limit on the number of filter coefficients which can be passed, which depending on the signal is between 6,000-8,000 coefficients, so shorter tones will ensure fewer coefficients will be generated. The tones were also designed with sharp peaks at their minimum and maximum frequency so the specific range of the filter matrix can be assigned. For example, for the full sweep tones, sharp peaks at 20 Hz and 20 kHz indicate the full range needed for the filter matrix's length.

Technical Issues

Audio Sync

Our transfer function was developed with the use of two audio signals: the test tone and the received tone. Obtaining identically sized audio signals proved to be a pressing non-trivial issue. The latency between running a line of code to start a recording and the microphone starting to record was the source of the slight discrepancy in audio signal sizes. In order for the filter generation method's matrix algebra to work properly, the two arrays must be the exact same length, as even a difference in one element would prevent it from properly running.

Fourier Transform

Another technical issue was our Fourier transform calculation. The direct applications of the Fourier transform lead to a doubling of the frequency domain signal **as** shown in Figure 4. These issues were resolved with the development of a specific type of filter generation tone. In addition to the tones described above, brief 20 Hz and 20 KHz tones were appended onto the generation tone as shown by the sharp edges in figure 5. The purpose of these frequencies was to extend the length of the signal to ensure the relevant ranges were recorded and to find the elements that would provide the coefficients of a useful filter. This ensured that the arrays of original and recorded signals were identical and the doubled frequency signal was removed.

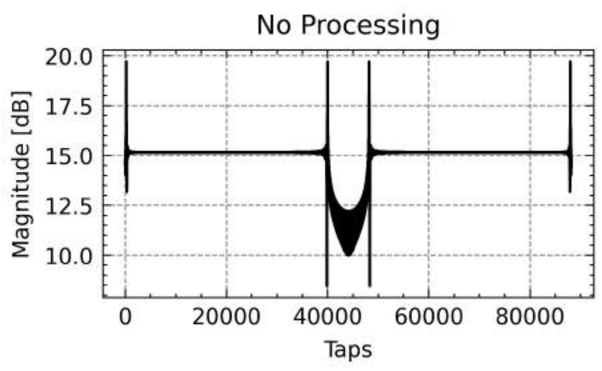


Figure 4: Signal w/ echo

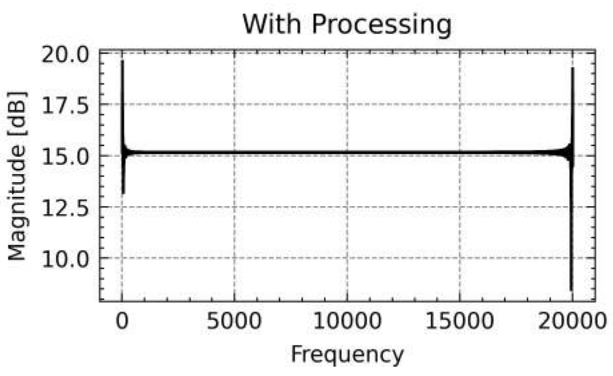


Figure 5: Processed signal w/ no echo

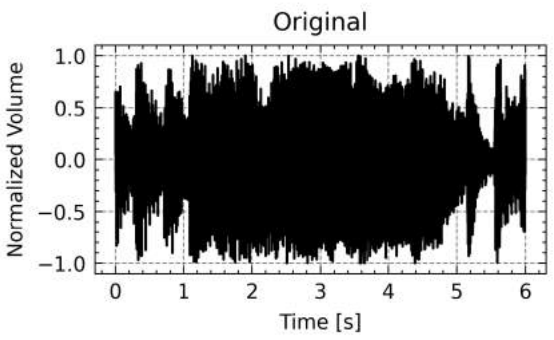
Teensy and Serial Communication

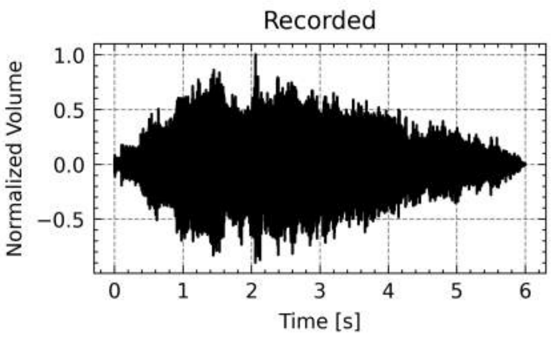
The use of a Teensy microcontroller also created unintentional complications with development. The default "filter_fir.cpp" has a hard encoded limit to how many coefficients a filter can contain and the Teensy has a coefficient limit based on its processing power. The first issue can be quickly resolved by making an edit to the "filter_fir.cpp" file; the second issue took several trials to find the upper limit. We were eventually able to determine that 8000 taps was the upper limit of coefficients that the Teensy could handle. At a 44.1 KHz sampling rate, this translates to about 0.18 seconds of audio.

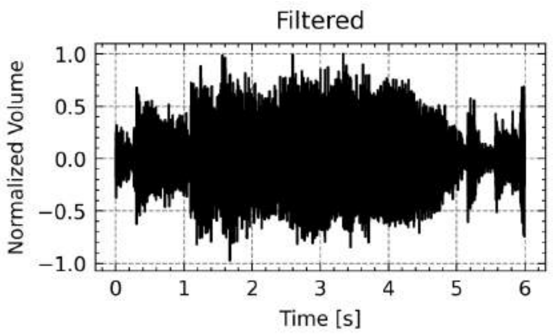
The current challenge is serial communication between the RP4 and the Teensy; this is due to two independent obstacles. The first is that within the Python3 “serial” library, the function “Serial”, which allows for sending information through serial connection, is no longer contained in the most recent version. The only solution that we found to this problem was copying an older version of the library to our RP4. The second obstacle is what is currently being resolved. We are sending our coefficients as a string with commas used as delimiters. We are currently trying to implement a function in the .ino file that allows us to read in this string and reformat it as an array. Once this development is completed, the iterative method will be finalized.

Final Results and Reflections

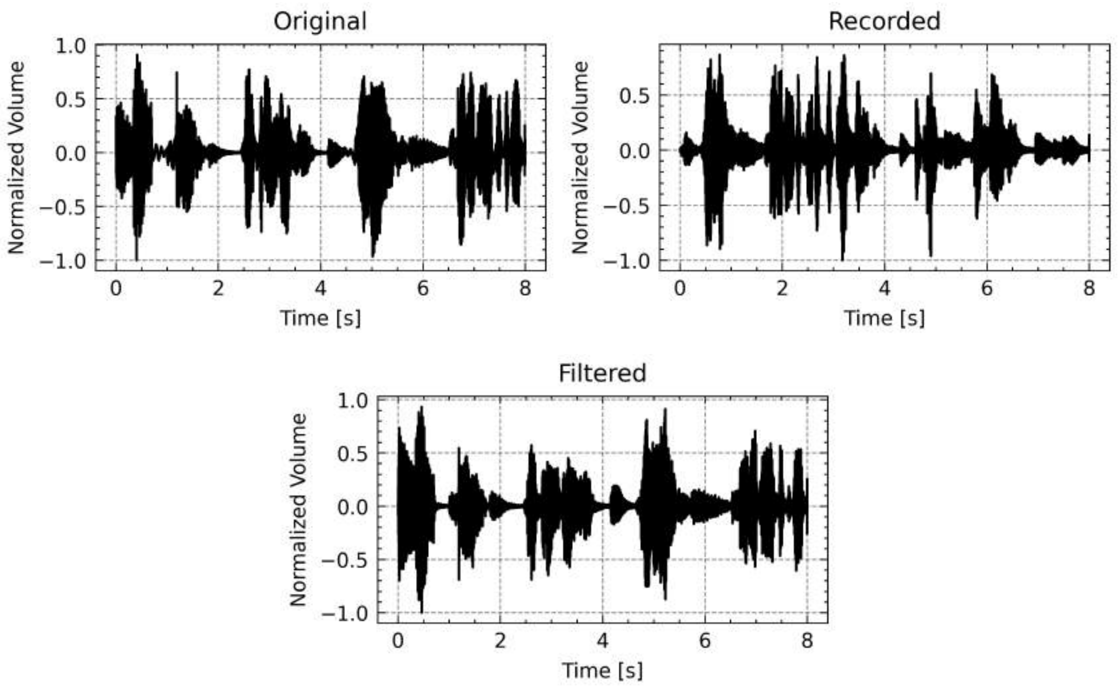
For the final test of our single iteration method, we applied a 10 ms white noise signal for filter generation. The generated filter was flashed to the Teensy and the Teensy was disconnected from the Raspberry Pi 4. A personal computer was connected through the Teensy to a 5.1 surround sound system and powered through a regular USB block. It was then tested on two separate signals: a contemporary pop song and a dialogue sample with background noise. Below are plots depicting the results of these two tests, showing the input signal and the filtered output.







Figures 6, 7 and 8: original pop song signal, received signal, and filtered signal respectively



Figures 9, 10 and 11: original dialogue with background signal, received signal, and filtered signal respectively

Our current method produces poor sound quality. The filtered audio is muffled and has extra noise compared to the original audio from the speaker. This is likely due to our single filter generation method. If we implemented our iterative method, the generated filter would be more stable and thus produce better sound quality.

Future Work

Given more time, there are several things we would work to implement going forward. We would finish implementing the iterative method for filter generation, write a program that better accounts for varying microphone frequency responses and write a program that allows for better control over reverberation and echo.

Due to time constraints and issues with serial communication, we were unable to get the iterative system fully working. A fully implemented iterative approach would allow for the system to dynamically determine which filter to apply based on applying a filter and generating output, applying another filter and generating output, comparing the two and moving forward with the more accurate output. To implement this, it would be necessary to have serial communications between the input device communicator and the teesny. This implementation would require a better understanding of flow control to the Teensy via serial to send coefficients and outputs back and forth quickly.

A revelation we had towards the end of our time working on this project was the effect the microphone quality has on the accuracy of the system. With a high quality microphone it is easy to get a clear input test tone. With a lower quality microphone it is not necessarily true that our system would perform at a high accuracy level due to inaccuracies in the test tone we would receive. This complicates the use case of the system as different phones have different microphones and few phones have a high quality audio processing microphone. It would be greatly beneficial to test the system with different microphones and compare results between the different setups. This would allow us to determine if this is a relevant issue that we need to account for. If the test proves that there is a significant difference in the accuracy of our system with varying microphones, the next step would be to implement a program that accounts for the varying frequency responses on the microphones. By programming this into the system, the accuracy would be unchanged independent of the microphone on the device that was used.

This semester we were only able to alter the frequency response of the speaker. We would like to have better control over reverberation and echo in the future. Similarly to frequency response, these two factors have an impact on the clarity of sound. An audio system cannot be truly optimized unless all three of these factors are accounted for and handled appropriately. By implementing mechanisms that will allow a way to control the echo and reverberation of any noise output onto the speaker, we would be able to produce what is essentially a completely optimal sound system.

References

B. Pomerantz, “Using test tones to set amplifier gain,” *Crutchfield*. [Online]. Available: https://www.crutchfield.com/S-RLxfeRMp0fY/learn/setting-amplifier-gain.html. [Accessed: 12-Mar-2021].

D. Asante, “Simple Ways To Test Your Room's Frequency Response,” *TechMuze Academy*. [Online]. Available: https://techmuzeacademy.com/simple-ways-to-test-your-rooms-frequency-response/. [Accessed: 29-Mar-2021].

J. Longman, “Understanding Speaker Frequency Response,” *AudioReputation*, 21-Jan-2021. [Online]. Available: https://www.audioreputation.com/understanding-speaker-frequency-response/. [Accessed: 02-Apr-2021].

K. Nunez, “What Is Pink Noise and How Does It Compare with Other Sonic Hues?,” *Healthline*, 21-Jun-2019. [Online]. Available: https://www.healthline.com/health/pink-noise-sleep. [Accessed: 02-Apr-2021].

S. K. Davis, “Types of Audio Processing: A Primer –,” *HARMAN Professional Solutions Insights*, 13-Mar-2017. [Online]. Available: https://pro.harman.com/insights/enterprise/education/types-of-audio-processing-a-primer/. [Accessed: 10-Mar-2021].

J. Smoot, “Understanding Audio Frequency Range in Audio Design,” *CUI Devices*, 04-Jun-2020. [Online]. Available: https://www.cuidevices.com/blog/understanding-audio-frequency-range-in-audio-design. [Accessed: 12-Mar-2021].

A. Santos, “THX releases iOS app for calibrating your home theater's visuals and audio,” *Engadget*, 20-Feb-2020. [Online]. Available: https://www.engadget.com/2013-01-30-thx-tune-up-calibration-app-ios.html. [Accessed: 09-Mar-2021].

Audyssey Laboratories. [Online]. Available: https://audyssey.com/. [Accessed: 09-Mar-2021].

“BEFORE USING THE UNIT,” *Optimizing the speaker settings automatically (YPAO)*. [Online]. Available: https://manual.yamaha.com/av/18/rxv685/en-US/311793035.html#:~:text=The%20Yamaha%20Parametric%20room%20Acoustic,parameters%2C%20to%20suit%20your%20room. [Accessed: 09-Mar-2021].

S. Guttenberg, “Get better sound from your AV receiver using your smartphone,” *CNET*, 25-Feb-2017. [Online]. Available: https://www.cnet.com/how-to/how-to-improve-your-av-receivers-sound-quality/. [Accessed: 09-Mar-2021].