EFFICIENT RASPBERRY PI IMPLEMENTATION OF A SISO ACTIVE CONTROL SYSTEM USING PYTHON

Charlie House, Jordan Cheer

Institute of Sound and Vibration Research, University of Southampton

ABSTRACT

Active control systems are used in many applications to reduce a disturbance signal with the addition of control sources. Commonly, these systems are run on dedicated Digital Signal Processing (DSP) hardware, designed to run such tasks efficiently and with minimal system latency. Whilst these dedicated platforms are effective, they can be costly and are therefore limiting in many applications. This paper presents a Python implementation of an adaptive Single Input Single Output (SISO) feedforward active control system, which is sufficiently efficient to run in realtime on the low-cost Raspberry Pi micro-computing platform. The use of the popular Python programming language makes the system accessible to non-experts, and the wide range of modules available for Python open up significant opportunities for integration with other systems.

Index Terms— Active Noise Control, Python, Raspberry Pi, Block-LMS

1. INTRODUCTION

Recent advancements in computer design have resulted in fairly high-speed processor boards (such as the Raspberry-Pi) becoming readily available at a very low cost. This has changed the way computers are seen by society, with many people now seeing micro-computers as a practical and affordable way of solving every day problems. Increased use of open-source software sharing sites, such as GitHub, have enabled novice programmers to readily gain access to high quality code, allowing them to write more complex software whilst developing their skills. The combination of these two developments has caused a birth in 'smart-home' tech, with companies such as Apple [1], Amazon [2], and Google [3] all developing platforms to allow low-cost consumer electronic devices to communicate with one another. This phenomena, known as the Internet-Of-Things (IOT), enables non-experts to develop custom software and hardware to do all manner of basic tasks based on other sensors; for example turning the lights on when someone's phone GPS senses they are near their house, or setting the television to mute when they receive a phone call.

Active control systems are also becoming increasingly common, with many companies selling low-cost consumer products incorporating an active element [4, 5]. Currently, however, there is no simple method to easily incorporate active control technology into a network of connected devices within the home. These systems are usually developed on costly algorithm-prototyping platforms such as dSPACE [6], before the production versions are implemented on a dedicated DSP board, such as Field Programmable Gate Array (FPGA) boards [7, 8]. Whilst there are toolboxes available to compile MATLAB code onto these boards [9], optimal performance is usually obtained by manually coding the algorithms in a low-level programming language such as C++ [8].

This paper will investigate an implementation of a single-channel, single-frequency feedforward active control system written in Python, and capable of running on a Raspberry-Pi. Such a system would be very cheap to implement, and the simplicity of the Python programming language will allow easy integration with other smart devices or systems. Initially the Block-FxLMS algorithm will be discussed, and the effect of frame size on convergence time and computational load will be investigated. Secondly, the hardware and software required to develop the active control system will be discussed, before the performance of the system is analysed.

2. THE BLOCK-FXLMS ALGORITHM

The FxLMS algorithm is an adaptive method used to minimise a given error function by adjusting a set of filter weights. It is commonly used in many Active Control applications, where the disturbance signal is time variant and therefore a control signal must be calculated in real-time, based on a measured error signal. In this case, the reference signal $\mathbf{x}$ is filtered by a set of weights $\mathbf{w}$, which are dynamically adjusted by the FxLMS algorithm to minimise a given error signal $\mathbf{e}$ as shown in Figure 1 where $\mathbf{d}$ is the primary disturbance signal.

The filter coefficients $\mathbf{w}$ begin from a pre-determined initial state (usually all set to 0), and are then updated, at each sample, by [10]

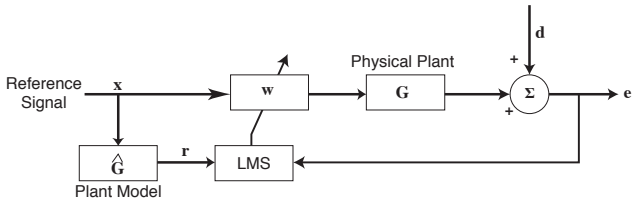

Fig. 1. A schematic diagram of the adaptive feedforward FxLMS algorithm.

$$\mathbf{w}(n+1) = \mathbf{w}(n) - \alpha \mathbf{e}(n)\mathbf{r}(n) \tag{1}$$

where $\mathbf{e}(n)$ is the error at the current step and $\mathbf{x}(n)$ is the reference signal at the current step.

To execute this algorithm in real-time on a single-sample DSP board, the controller has one sample period, $T_s = \dfrac{1}{f_s}$ seconds to read one sample of error signal from the ADC, calculate the updated control filter weights, generate the filtered reference signal and then write this signal to the DAC. This is fairly simple to achieve with low-level programming languages such as C++, where the code is compiled to efficiently run on a fast processor, however with non-compiled, and higher level, code such as Python, or when running the control algorithm on a low-power processor such as the Raspberry Pi, this is more challenging. For this reason, many real-time processing tasks work on frames of data rather than individual samples [11, 12].

With a frame based processing architecture, the processor reads N samples from the ADC and stores the data in a buffer. All processing steps are then run on this vector of data, before each sample is written to the DAC at the correct time to achieve smooth and continuous playback. While a per sample based algorithm has T_s seconds in which to carry out the above tasks, a frame based algorithm has NT_s seconds, where N is the length of the frame, giving the processor an increased window within which to complete the necessary processing steps. The Block-FxLMS algorithm is a modification of the standard FxLMS algorithm outlined above, designed to update the filter weights every N samples. This has significant advantages in computational cost, however, the controller has to wait for a longer duration before receiving an updated error signal, resulting in a slower convergence time. The update equation for the Block-FxLMS algorithm with a frame-size of N samples is [12, 13, 14]

$$\mathbf{w}(n+N) = \mathbf{w}(n) - \frac{\alpha}{N}\sum_{i=0}^{N-1} \mathbf{e}(i)\mathbf{r}(i) \tag{2}$$

where $1 < i < N$ is the sample index within the frame.

2.1. Effect of Frame Size

To investigate the effect of the frame size on both the computational load and convergence time, an offline time-domain model of the Block-FxLMS algorithm has been implemented. Specifically, a Single-Input-Single-Output (SISO) feedforward tonal controller, running at $f_s = 48$kHz, with a disturbance at 250Hz has been implemented. The plant response used in the model was defined as the response measured between a loudspeaker and a microphone in a reverberant environment at 250Hz. All of the results presented in this section have been computed using MATLAB R2017a running on a 2013 MacBook Pro with a dual-core 2.8 GHz Intel Core i7 processor.

An investigation has initially been conduced into the effect of the frame size on the computational demand. The frame size has been increased from 1 sample to 1024 samples, increasing in powers of 2, and in each case the modelled Block-FxLMS algorithm has been run for 30sec and the required CPU-time has been measured. This has been carried out 10 times for each frame size and the results averaged, to reduce the influence of processor demand from other unrelated tasks. The average compute time is shown in Figure 2 for increasing frame size. From these results it can be seen that increasing the frame size from 1 to around 128 results in a significant reduction in the computational load, whilst further increases in the frame size offer diminishing returns.

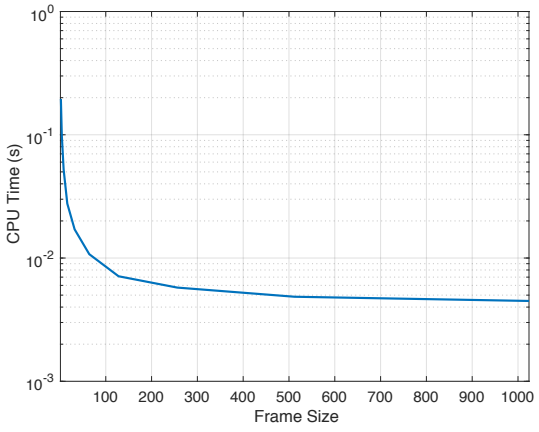

Fig. 2. The effect of frame size on the CPU load of the Block-FxLMS algorithm.

The effect of frame size on the convergence time of the FxLMS algorithm has also been investigated. The convergence time has been defined as the time taken for the algorithm to achieve a 20 dB reduction in the error signal, and this has been measured for a range of frame sizes. In each case, the convergence coefficient α was set to achieve the fastest possible convergence speed for the given frame size. The results are presented in Figure 3, with the convergence

time normalised to the convergence time for a frame-size of 1. From these results it can be seen that increasing the frame size results in a significant increase in the convergence time.

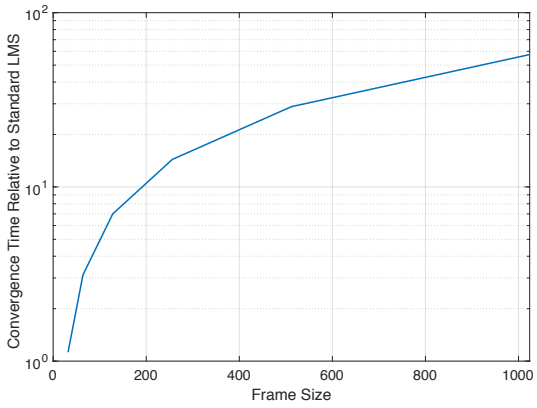

Fig. 3. The effect of frame size on the convergence time of the Block-FxLMS algorithm.

From the results presented in Figures 2 and 3, it is clear that the frame-size can be used to reach a trade-off between computational load and convergence time. For a particular hardware setup, it can also be used to achieve the minimum convergence time whilst ensuring data-overrun errors do not occur [15]. The frame-size that gives the minimum convergence time whilst providing sufficient computational time can be found by comparing the results presented in Figure 2 to the maximum available processing time per frame NT_s seconds, and this is shown in Figure 4. It can be seen for this MATLAB implementation of the Block-FxLMS algorithm, running on the specific hardware detailed previously, the frame-size should be at least 512 samples to maintain computational stability.

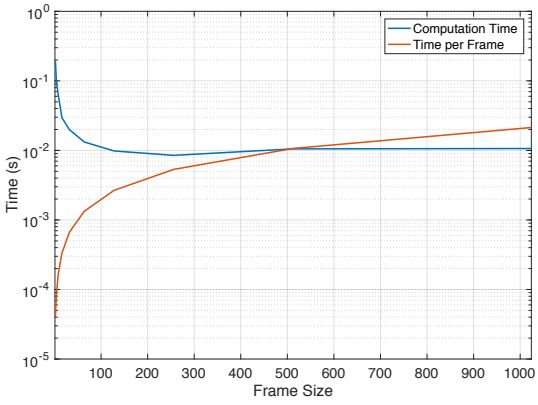

Fig. 4. A comparison between the required computational time and the available processing time for a range of block sizes.

3. PYTHON IMPLEMENTATION OF A SISO FEEDFORWARD CONTROLLER

A real-time implementation of the Block-FxLMS algorithm discussed above will now be introduced. An adaptive feedforward single-tone controller is designed, and the performance of an Active Noise Control system using this controller is discussed.

A Raspberry-Pi 3B, running the Raspbian Linux distribution, is connected to a Focusrite 2i4 USB-soundcard acting as both an ADC and a DAC. The PyAudio package within Python-3 is used to handle the audio IO tasks, with the Numpy package being used to implement Equation 2, with the audio samples being represented as 32bit floating point values. The control algorithm is run in a callback function; being called with each frame of input data as shown in Figure 5.

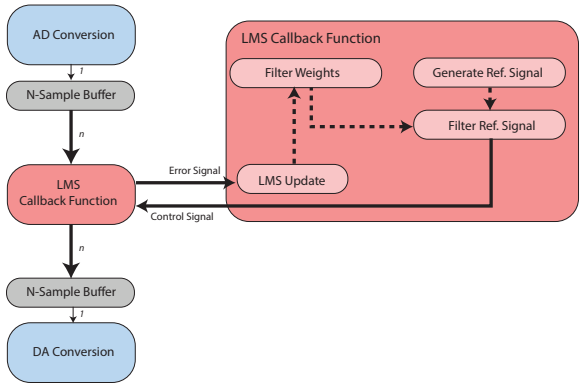

Fig. 5. A schematic diagram of the Python implementation of the Block-FxLMS control algorithm.

The use of a callback function ensures the CPU runs the control algorithm in a separate processor thread to the main program. This allows the computational overheads for the realtime stream (input/output buffers, PortAudio drivers for the ADC/DAC etc) to run separately to the control algorithm, increasing computational efficiency and therefore allowing for reduced frame sizes or increased sampling frequencies.

To investigate the performance of the system discussed above, an experimental setup has been created in a laboratory. A loudspeaker emitting a tone was used as the primary disturbance, whilst a low-budget microphone and secondary loudspeaker were connected to the Focursite sound-card and the Raspberry Pi to form the ANC system, as shown in Figure 6. The system was run at $f_s = 44.1$kHz, with a frame size of 1024 samples. Equation 2 was adapted to include a leakage term β, as shown in Equation 3, which has been set to a value of $\beta = 1 \times 10^{-4}$, and provides regularisation to the system to

increase robustness to error.

$$\mathbf{w}(n+N) = (1-\beta\alpha)\,\mathbf{w}(n) - \frac{\alpha}{N}\sum_{i=1}^{N}\mathbf{e}(i)\mathbf{r}(i) \qquad (3)$$

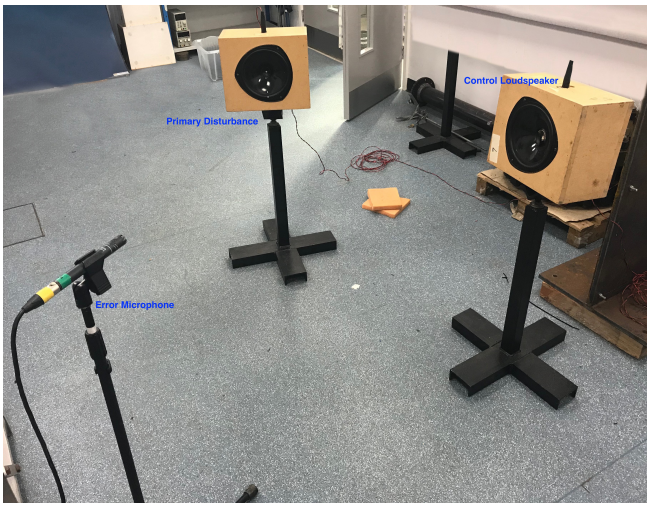

Fig. 6. Experimental laboratory setup.

3.1. Results

The above experimental setup was used to assess the performance of the active control system at 100Hz, 150Hz and 250Hz. In each case, the controller was enabled after 2 seconds, with the error signal being recorded for a further 28s to measure the convergence. The achieved attenuation in the error signal at each frequency is presented in Table 1, and the convergence plots are given in Figure 7, with a dashed line indicating the time at which the controller was enabled.

Frequency (Hz)	Attenuation (dB)
100	21.8
150	13.3
250	30.4

Table 1. Measured attenuation performance of the Active Control system.

It can be seen from Figure 7 and Table 1 that the controller is able to achieve significant reductions in the error signal at all frequencies considered, effectively reducing the disturbance signal to the background noise level in each case, as expected for a SISO controller.

4. CONCLUSIONS

There is a demand for low-cost simple active noise control systems to support the growing trend for 'smart-devices' and

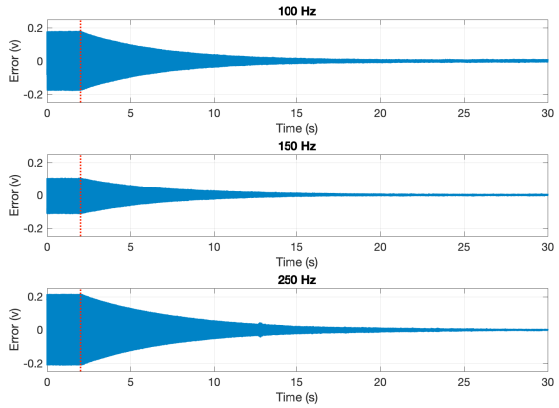

Fig. 7. Time history of the error signal when the active control system is run at three different frequencies. The dashed line indicates the time at which the controller was enabled.

the Internet of Things. The Block-FxLMS algorithm can be used in this case, which achieves a significant reduction in the computational demand when compared to the standard FxLMS algorithm. The Block-FxLMS algorithm has been discussed, and the effect of frame size on the convergence time and processor load has been investigated. It was found that increasing the frame size significantly decreases the required computation time, but in doing so sacrifices convergence speed.

An efficient implementation of the Block-FxLMS algorithm has been implemented in Python, and a proof-of-concept system has been created using a Raspberry Pi and other low-cost hardware. The performance of this system has been investigated in a laboratory setup, and significant reductions in the acoustic error signal were achieved.

Whilst the system discussed here is only a single-channel tonal controller, the software structure outlined in Figure 5 could be adapted relatively easily to utilise multiple error sensors and control sources, potentially enabling global control of a soundfield to be achieved [16]. There is also an interest in implementing a broadband controller, however it has been noted that the latency of the USB soundcard and the large-block size needed to run on a Raspberry Pi may make this non-trivial. A frequency domain implementation of the Block-FxLMS algorithm may be beneficial in this instance, as it could achieve reductions in the computational demand of many orders of magnitude [17, 18, 19].

All python scripts discussed in this paper can be downloaded and used free of charge from GitHub at `https://github.com/CharlieHouse/RPi_SISO_ANC`.

5. REFERENCES

[1] Apple, "Apple HomeKit." [Online]. Available: https://www.apple.com/uk/ios/home/

[2] Amazon, "Amazon Alexa." [Online]. Available: https://www.amazon.com/b?node=16067214011

[3] Google, "Google Home." [Online]. Available: https://store.google.com/gb/product/google{_}home

[4] Bose Corporation, "Bose Noise Cancelling Headphones." [Online]. Available: https://www.bose.co.uk/en{_}gb/products/headphones/noise{_}cancelling{_}headphones.html

[5] Sony Europe, "Sony Wireless Noise Cancelling Headphones." [Online]. Available: https://www.sony.co.uk/electronics/headband-headphones/wh-1000xm2

[6] DSPACE, "dSPACE Prototyping Systems." [Online]. Available: https://www.dspace.com/en/inc/home/products/systems/functp.cfm

[7] C. Dick and F. Harris, "FPGA signal processing using Sigma-Delta modulation," *IEEE Signal Processing Magazine*, vol. 17, no. 1, pp. 20–35, 2000.

[8] R. Woods, J. McAllister, Y. Yi, and G. Lightbody, *FPGA-based Implementation of Signal Processing Systems*. John Wiley & Sons, 2008.

[9] Z. German-Sallo, "Signal Processing using FPGA Structures," *Procedia Technology*, vol. 12, pp. 112–118, 2014. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S2212017313006488

[10] S. J. Elliott, *Signal Processing for Active Control*. Academic Press, 2001.

[11] Burrus, "Block Implementation of Digital Filters," vol. c, 1971.

[12] G. Clark, S. Mitra, and S. Parker, "Block implementation of adaptive digital filters," *IEEE Transactions on Circuits and Systems*, vol. 28, no. 6, pp. 584–592, 1981. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1085018

[13] P. Vikram Kumar, K. Prabhu, and D. Das, "Block filtered-s least mean square algorithm for active control of non-linear noise systems," *IET Signal Processing*, vol. 4, no. 2, p. 168, 2010. [Online]. Available: http://digital-library.theiet.org/content/journals/10.1049/iet-spr.2008.0157

[14] E. Ferrara, "Fast implementations of LMS adaptive filters," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 28, no. 4, pp. 474–475, 1980. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1163432

[15] L. Hsieh and S. Wood, "Performance Analysis of Time Domain Block LMS Algorithms," *IEEE*, 1993.

[16] S. C. Douglas, "Fast implementations of the filtered-X LMS and LMS algorithms for multichannel active noise control," *IEEE Transactions on Speech and Audio Processing*, vol. 7, no. 4, pp. 454–465, 1999.

[17] D. P. Das, G. Panda, and D. K. Nayak, "Development of Frequency Domain Block Filtered-s LMS (FBFSLMS) Algorithm for Active Noise Control System," *Spectrum*, pp. 289–292, 2006.

[18] M. Chakraborty and R. Shaik, "The block LMS algorithm and its FFT based fast implementation - New efficient realization using block floating point arithmetic," *European Signal Processing Conference*, vol. 2, no. Eusipco, pp. 0–4, 2006.

[19] N. K. Rout, D. P. Das, and G. Panda, "Computationally efficient algorithm for high sampling-frequency operation of active noise control," *Mechanical Systems and Signal Processing*, vol. 56, pp. 302–319, 2015. [Online]. Available: http://dx.doi.org/10.1016/j.ymssp.2014.10.009