***DSP Education Kit***

**LAB 6**

**Adaptive Filters**

**Issue 1.0**

Contents

[1 Introduction 1](#_Toc29196168)

[1.1 Lab overview 1](#_Toc29196169)

[2 Requirements 1](#_Toc29196170)

[3 Adaptive Filter Using C Code 1](#_Toc29196171)

[4 Adaptive FIR Filter for Noise Cancellation Using External Inputs 6](#_Toc29196172)

[5 Normalized Least Mean Squares Algorithm 7](#_Toc29196173)

[6 Adaptive FIR Filter for System Identification of IIR Filter 8](#_Toc29196174)

[6.1 Exercise 9](#_Toc29196175)

[7 Conclusions 10](#_Toc29196176)

# Introduction

## Lab overview

The examples in these exercise concern different applications of an adaptive FIR filter using the Least Mean Squares (LMS) algorithm.

# Requirements

To carry out this lab, you will need:

* An STM32F746G Discovery board
* A PC running Keil MDK-Arm
* MATLAB
* An oscilloscope
* Suitable connecting cables
* An audio frequency signal generator

# Adaptive Filter Using C Code

This example applies the Least Mean Square (LMS) algorithm, coded in C, to pre-determined input and desired output signals (sequences). It illustrates the following steps in the adaptation process using the adaptive structure shown in Figure 1.

1. Obtain new input values *x*(*n*) and desired output sample *d*(*n*).
2. Compute the output of the adaptive FIR filter *y*(*n*) using equation (1).
3. Compute the instantaneous error signal *e*(*n*) using equation (2).
4. Update each of the adaptive FIR filter’s coefficients (weights) using equation (3). This is the LMS approximation of the iterative steepest descent algorithm.
5. Update the contents of the delay line containing *N* previous input samples.

These steps are repeated at every sampling instant.

 (1)

 (2)

 (3)



Figure 1: Block diagram of adaptive filter implemented by program stm32f7\_adaptive.c

The following code snippet shows the program stm32f7\_adaptive.c that implements the LMS algorithm for the adaptive filter structure shown in Figure 1.

The desired output signal used in program stm32f7\_adaptive.c is

 (4)

and the input signal is

 (5)

The learning rate, number of filter coefficients, and number of sample instants simulated by the program are 0.01, 21, and 64, respectively.

// stm32f7\_adaptive.c

#include "stm32f7\_wm8994\_init.h"

#include "stm32f7\_display.h"

#define SOURCE\_FILE\_NAME "stm32f7\_adaptive.c"

#define BETA 0.01f // learning rate

#define N 21 // number of filter coeffs

#define NUM\_ITERS 64 // number of iterations

float32\_t desired[NUM\_ITERS]; // storage for results

float32\_t y\_out[NUM\_ITERS];

float32\_t error[NUM\_ITERS];

float32\_t w[N+1] = {0.0}; // adaptive filter weights

float32\_t x[N+1] = {0.0}; // adaptive filter delay line

int i, t;

float32\_t d, y, e;

int main()

{

for (t = 0; t < NUM\_ITERS; t++)

{

x[0] = sin(2\*PI\*t/8); // get new input sample

d = cos(2\*PI\*t/8); // get new desired output

y = 0; // compute filter output

for (i = 0; i <= N; i++)

y += (w[i]\*x[i]);

e = d - y; // compute error

for (i = N; i >= 0; i--)

{

w[i] += (BETA\*e\*x[i]); // update filter weights

if (i != 0)

x[i] = x[i-1]; // shift data in delay line

}

desired[t] = d; // store results

y\_out[t] = y;

error[t] = e;

}

stm32f7\_LCD\_init(0, SOURCE\_FILE\_NAME, GRAPH);

while(1)

{

plotWave(desired, NUM\_ITERS, 0, 0);

proceed\_statement();

plotWave(y\_out, NUM\_ITERS, 0, 0);

proceed\_statement();

plotWave(error, NUM\_ITERS, 0, 0);

proceed\_statement();

}

}

Now, run the program stm32f7\_adaptive and observe its outputs by following these steps:

1. Build and run program stm32f7\_adaptive.c. The program stores the desired output, output and error signals for  in arrays desired, y\_out, and error respectively. The arrays are of type float32\_t.
2. By pressing the blue user pushbutton on the Discovery board, you can cycle through graphs on the LCD of the first 64 sample values of desired, y\_out, and error.
3. Halt the program and save the contents of these arrays to data files by entering

SAVE desired.dat <start address>, <start address + 0x100>

SAVE y\_out.dat <start address>, <start address + 0x100>

SAVE error.dat <start address>, <start address + 0x100>

in the *Command* window in the *MDK-Arm* debugger. Use the *Memory* window to find the start addresses of the arrays desired, y\_out, and error.

1. Plot the contents of each of the data files using MATLAB function STM32F7\_BAR\_real(). The filter output should have converged to the desired output and the error should have decreased over the 64 sample instants simulated as shown in the following figures.



Figure 2: Desired output desired , simulated using program stm32f7\_adaptive.c

.

Figure 3: Adaptive filter output y\_out, simulated using program stm32f7\_adaptive.c



Figure 4: Error signal error, simulated using program stm32f7\_adaptive.c

1. **Repeat the experiment using a learning rate (beta) of 0.02 and verify that convergence is faster.**

Program stm32f7\_adaptive.c is an extremely simplistic demonstration of an adaptive filter. It is intended to introduce the relationships between input, output, desired output and error signals, and the role of the learning rate, and to illustrate how simple it can be to implement the LMS algorithm.

# Adaptive FIR Filter for Noise Cancellation Using External Inputs

Program stm32f7\_noise\_cancellation\_intr.c requires two external inputs, a desired signal and a reference noise signal to be input to left and right channels, respectively. Test input signals are provided in file speechnoise.wav. This may be played through a PC soundcard and input to the LINE IN socket on the audio card via a stereo 3.5 mm jack plug to 3.5 mm jack plug cable. speechnoise.wav comprises pseudorandom noise on the left channel and speech on the right channel.

Figure 5 shows the program in block diagram form. Within the program, a primary noise signal, correlated to the reference noise signal input on the left channel, is formed by passing the reference noise through an IIR filter. The primary noise signal is added to the desired signal (speech) input on the right channel.

adaptive

filter

**+**

**+**

**-**

**+**

signal

refnoise

signoise

IIR

filter

LINE IN L

LINE IN R

HP OUT L

HP OUT R

error

Figure 5: Block diagram representation of program stm32f7\_noise\_cancellation\_intr.c

Build and run the program and test it using file speechnoise.wav. As adaptation takes place, the output on the left channel of HEADPHONE OUT should gradually change from speech plus noise to speech only. You may need to adjust the volume at which you play the file speechnoise.wav. If the input signals are too quiet, then the adaptation may be very slow.

While the program is running, use the blue user pushbutton to toggle between graphs on the LCD showing the adaptive filter coefficients (the impulse response of the adaptive filter) and the magnitude of their Fast Fourier Transform (FFT).

After adaptation has taken place, and the program has been halted, the 256 coefficients of the adaptive FIR filter, firCoeffs32, may be saved to a data file by typing:

**SAVE <filename> <start address>, <end address>**,

where start address is the address of array firCoeffs32 and end address is equal to start address + 0x400, and plotted using the MATLAB function stm32f7\_logfft(). The filter coefficients should reveal the impulse and magnitude frequency responses of the IIR filter implemented by the program and shown at the left-hand side of Figure 5. The characteristics of the IIR filter are determined by the coefficients in header file bilinear.h. You can substitute different coefficients by including, for example, header file elliptic\_bp.h.

# Normalized Least Mean Squares Algorithm

In the previous example, you may have noticed that the rate of adaptation of the system could be influenced by the amplitudes of the signals involved. This effect can be reduced by using the *normalized* LMS algorithm–the steps involved are summarized below.

1. Obtain new input and desired output sample values *x*(*n*) and *d*(*n*).
2. Compute the output of the adaptive FIR filter *y*(*n*) using equation (1).
3. Compute the instantaneous error signal *e*(*n*) using equation (2).
4. Compute the instantaneous energy, *energy*(*n*) of the values stored in the filter delay line (input buffer) *x*, using equation (6)
5. Update each of the adaptive FIR filter’s coefficients (weights) using equation (7).
6. Update the contents of the delay line containing *N* previous input samples.

These steps are repeated at every sampling instant.

 (6)

 (7)

Program stm32f7\_noise\_cancellation\_norm\_CMSIS\_intr.c is a very slightly modified version of program stm32f7\_noise\_cancellation\_CMSIS\_intr.c that implements the normalised LMS algorithm.

Program stm32f7\_noise\_cancellation\_norm\_CMSIS\_intr.c makes use of CMSIS library function arm\_lms\_norm\_f32() in place of function arm\_lms\_f32() and uses a far larger learning rate, beta.

You should be able to verify that program stm32f7\_noise\_cancellation\_norm\_CMSIS\_intr.c is relatively insensitive to the volume at which the test file speechnoise.wav is played.

# Adaptive FIR Filter for System Identification of IIR Filter

Program stm32f7\_iirsosadapt\_intr.c uses an adaptive FIR filter configured for system identification of an IIR filter as shown in Figure 6.

adaptive

filter

+

-

IIR

filter

PRBS

HP OUT R

HP OUT L

error

adapt\_out

Figure 6: Block diagram representation of program stm32f7\_iirsosadapt\_intr.c

Adaptation takes place in real-time while the same Pseudorandom Binary Sequence (PRBS) is input to both filters. You can listen to, or watch on an oscilloscope, the output of the adaptive FIR filter adapt\_out and/or the difference between the outputs of the two filters error. As the FIR filter assumes the characteristics of the IIR filter, the variance of the error signal decreases.

For the purposes of appreciating the behavior of the adaptive filter, its rate of adaptation beta has deliberately been set very low.

While the program is running, use the blue user pushbutton to toggle between graphs on the LCD showing the adaptive filter coefficients (the impulse response of the adaptive filter) and the magnitude of their FFT.

After adaptation has taken place, and the program has been halted, the 256 coefficients of the adaptive FIR filter, w, may be saved to a data file by typing:

**SAVE <filename> <start address>, <end address>**,

where start address is the address of array firCoeffs32 and end address is equal to start address + 0x400, and plotted using the MATLAB function stm32f7\_logfft(). The filter coefficients should reveal the impulse and magnitude frequency responses of the IIR.

This example shares many similarities with the noise cancellation example. Both use an adaptive filter configured for system identification. However, in the case of noise cancellation, the output signal of interest is the error between the desired output and the output of the adaptive filter. On the other hand, in this example, the interest might be said to lie in the output of the adaptive filter or in its coefficients. In both examples, an FIR filter is adapted so as to take on the characteristics of an IIR filter.

The characteristics of the IIR filter in this example are determined by the IIR filter coefficient header file elliptic.h that was generated using MATLAB’s fdatool (as described in Laboratory Manual 4).

## Exercise

**Plot the identified magnitude frequency response displayed using MATLAB on the axes below and comment on how it compares to the theoretical response predicted by fdatool.**





Figure 7: Theoretical magnitude frequency response of fourth-order, elliptic, IIR filter implemented using program stm32f7\_iirsosadapt\_intr.c

# Conclusions

This laboratory exercise has introduced the LMS and normalized LMS algorithms for adaptive FIR filters. Real-time implementations of noise cancellation and system identification have been demonstrated.