Institut EURECOM
2229, route des Crêtes, B.P. 193,
06904 Sophia Antipolis Cedex

Research Report N° 95-018

# The Fast Subsampled-Updating Fast Affine Projection (FSU FAP) Algorithm

Karim Maouche Dirk T.M. Slock

*December, 1994*

| | | |
|---|---|---|
| Telephone: | +33 93 00 26 26 | E-mail: |
| Karim Maouche: | +33 93 00 26 32 | maouche@eurecom.fr |
| Dirk T.M. Slock: | +33 93 00 26 06 | slock@eurecom.fr |
| Fax: | +33 93 00 26 27 | |

## Abstract

The Fast Affine Projection (FAP) Algorithm is the fast version of the AP algorithm which is a generalization of the well-known Normalized Least-Mean-Square (NLMS) algorithm. The AP algorithm shows performances that are near to those of the Recursive Least-Squares algorithms while its computational complexity is nearly the same as the LMS algorithm one. Moreover, recent research has enlightened the strong tracking ability of the AP algorithm, rendering it very interesting for adaptive systems that evolve within highly non-stationary environments. In order to reduce the $\mathcal{O}(2N)$ (N is the filter length) computational complexity of the FAP algorithm, we apply the Subsampled-Updating approach in which the filter estimate is provided at a subsampled rate, say every $M$ samples. Using the FFT technique when computing the products of vectors with Toeplitz matrices leads to the Fast Subsampled-Updating FAP (FSU FAP) algorithm which is mathematically equivalent to the AP algorithm. The FSU FAP algorithm shows a low computational complexity for relatively large filters while presenting good convergence and tracking performances. This makes the FSU FAP algorithm a challenging candidate for applications such as acoustic echo cancellation.

## Contents

## 1 Introduction

There are two major classes of adaptive algorithms: one is the Least-Mean-Square (LMS) algorithm which is based on a stochastic-gradient method [19] and the other is the Recursive Least-Squares (RLS) algorithm which minimizes a deterministic sum of squared errors [7]. The $2N$ low computational complexity of the LMS algorithm ($N$ is the filter order) renders it very popular. Nevertheless, the LMS algorithm shows a very slow convergence speed when the adaptive filter length is relatively large or when the input signal is correlated.
The RLS Algorithm shows a relatively important computational complexity of $\mathcal{O}(N^2)$. However, with Finite Impulse Response (FIR) filtering, consecutive regression vectors are related through a shift operation. This allows for the derivation of fast RLS algorithms with a complexity of $\mathcal{O}(N)$. The most popular between them is the Fast Transversal Filter (FTF) algorithm because of its lowest computational complexity which is equal to $7N$ [2]. Unfortunately, fast RLS versions suffer from round-off error accumulation that leads to numerical instability. Recently, a stabilized version of the FTF algorithm was derived which shows a computational complexity of $8N$ [15]. RLS algorithm is much more efficient than LMS algorithm since its superiority in convergence and tracking [4] but its complexity (even in the fast versions) disqualify it from being used in applications that are computationally demanding such as echo cancellation applications where the filter length turns around several hundreds in the case of mobile-radio communications and several thousands for teleconference communications.
Recently, the Fast Newton Transversal Filter (FNTF) algorithm has been derived that is intermediate to the two families [12]. The FNTF algorithm departs from the FTF algorithm and uses the approximation that when dealing with Auto-Regressive (AR) signals, the prediction part of the FTF algorithm can be limited to prediction filters and Kalman gain of length $P$, the order of the AR model. In fact, in the FNTF algorithm the inverse of the sample covariance matrix is approximated by a banded matrix of total bandwidth $2P+1$. This allows the reduction of the complexity to $\mathcal{O}(2N)$. The FNTF algorithm has been implemented successfully in a radio-mobile hands-free system [14]. It exhibited performances that are near to those of the RLS algorithm. Nevertheless, the performances of the FNTF algorithm become worst with longer filters than those used in mobile-radio communications.
The Affine Projection (AP) algorithm constitutes another intermediate to the LMS and RLS adaptive filtering algorithms families. The key ingredient of the AP algorithm is a $L$ dimensional projection whereas the NLMS algorithm is based on a one dimensional projection scheme. From this point of view, the AP algorithm constitutes a generalization to the Normalized LMS (NLMS) algorithm. The AP algorithm is known to converge faster than the NLMS even with white input signal. It has also a stronger tracking ability. The cost being the inversion at each iteration of a $L \times L$ sample covariance matrix that is estimated over a sliding rectangular window of length $N$. In [5], a fast version of the AP algorithm was derived by using the Sliding Window Covariance RLS (SWCRLS) algorithm, leading to the Fast AP (FAP) algorithm which computational complexity is $2N + \mathcal{O}(L)$. A perticularity of this fast version is the use of a pseudo-filter of length $N$ which allows the computation of the output error without updating the filter estimate.
When one deals with longer filters than those used in the mobile-radio context ($N > 256$), the FNTF, the FAP and even the NLMS algorithms can not be implemented because of todays technological limitations. In [16],[17],[10], we have pursued a way in order to reduce the complexity of RLS adaptive filtering algorithms. The approach consists of subsampling

the filter adaptation, i.e. the LS filter estimate is no longer provided every sample but every $M \geq 1$ samples (subsampling factor $M$). This strategy has led us to derive new RLS algorithms that are the FSU RLS, FSU SFTF and FSU FNTF algorithms which present a relatively small computational complexity when dealing with long FIR filters. Here, we apply the subsampled-updating strategy (SUS) to the FAP algorithm. In this approach, we keep the computations of $\mathcal{O}(L)$ as they are and compute the pseudo-filter and its corresponding output from the pseudo-filter that was available $M$ instants before. The SUS makes appear convolutions that are done with the Fast Fourier Transform (FFT) technique. This leads to a new algorithm with a reduced computational complexity, rendering it especially suited for adapting very long filters such as in the acoustic echo cancellation problem.
The rest of this report is organized as follows. In sections 2 and 3, we briefly recall the AP and FAP algorithms. In section 4, we apply the SUS to the FAP algorithm and derive the SU FAP algorithm. Section 5 deals with the fast computation of convolutions using the FFT in order to reduce the computational complexity of the SU FAP algorithm. This technique leads to the FSU FAP algorithm which is given in section 6. Finally, some concluding remarks are presented in section 7.
In order to formulate the problem and to fix notation, we shall first recall the AP algorithm and the FAP algorithm.

## 2 The AP Algorithm

An adaptive transversal filter $W_{N,k}$ forms a linear combination of $N$ consecutive input samples $\{x(i-n), n=0,\ldots,N-1\}$ to approximate (the negative of) the desired-response signal $d(i)$. The resulting error signal is given by

$$\epsilon_N(i|k) \;=\; d(i) + W_{N,k}\,X_N(i) \;=\; d(i) + \sum_{n=0}^{N-1} W_{N,k}^{n+1}\,x(i-n) \;\;, \tag{1}$$

where $X_N(i) = \left[x^H(i)\; x^H(i-1) \cdots x^H(i-N+1)\right]^H$ is the regression vector at time $i$, $W_{N,k}$ is the set of $N$ transversal filter coefficients $W_{N,k} = \left[W_{N,k}^1 \cdots W_{N,k}^N\right]$ and superscript ${}^H$ denotes Hermitian (complex conjugate) transpose.
The AP algorithm generalizes the NLMS algorithm and is given by

$$\left\{ \begin{array}{lcl} \epsilon_{N,L,k}^p & = & d_{L,k} + X_{N,L,k} W_{N,k-1}^H \\ \\ W_{N,k} & = & W_{N,k-1} - \mu_k \epsilon_{N,L,k}^{p\,H} \left(X_{N,L,k} X_{N,L,k}^H\right)^{-1} X_{N,L,k} \end{array} \right. \tag{2}$$

where

$$d_{L,k} = \begin{bmatrix} d_k^H \\ \vdots \\ d_{k-L+1}^H \end{bmatrix} \;\;,\;\; \epsilon_{N,L,k}^p = \begin{bmatrix} \epsilon_N^H(k|k-1) \\ \vdots \\ \epsilon_N^H(k-L+1|k-1) \end{bmatrix} \;\;, \tag{3}$$

$0 < \mu_k < 2$ is a step-size parameter called the relaxation factor and $X_{N,L,k}$ is the $L \times N$ Hankel input data matrix

$$X_{N,L,k} = \begin{bmatrix} X_N^H(k) \\ \vdots \\ X_N^H(k-L+1) \end{bmatrix} = \left[\, x_{L,k}\; x_{L,k-1} \cdots x_{L,k-N+1} \,\right] \;\;, \tag{4}$$

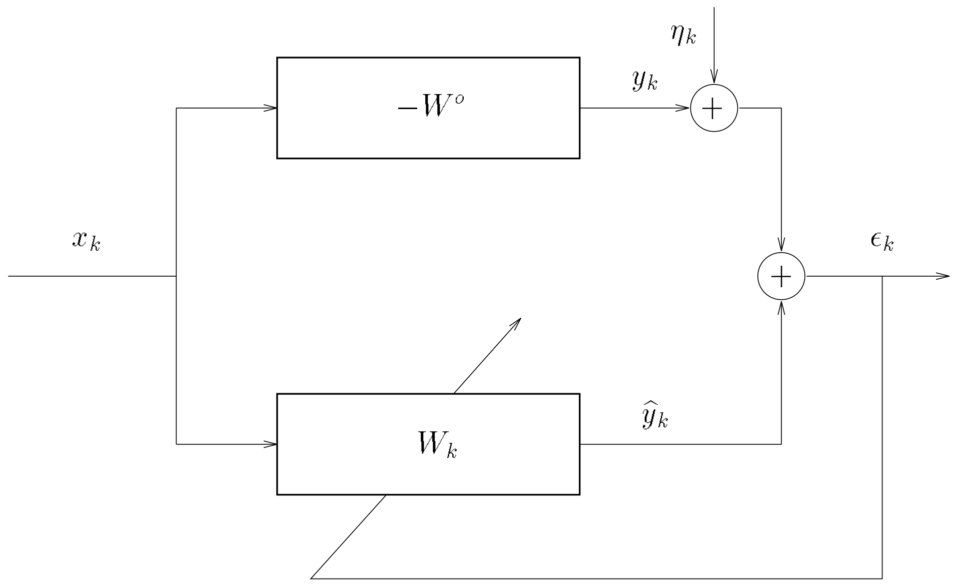

Figure 1: The identification scheme of an unknown plant.

with the complex-conjugate regressor vector

$$x_{L,k} = [\ x_k\ x_{k-1}\ \cdots\ x_{k-L+1}\ ]^H \quad . \tag{5}$$

The AP algorithm reduces to the NLMS algorithm when $L = 1$ and extends to the SWC RLS algorithm when $L = N$. This leads to a set of algorithms that are intermediate to the LMS and RLS adaptive filtering algorithms families.
In fact, it appears that the AP estimate filter at time $k$ is the solution to the following minimization problem

$$\min_{W_{N,k}} \|d_{L,k} + X_{N,L,k}W_{N,k}^H\|_{P_k}^2 + \|W_{N,k} - W_{N,k-1}\|^2 \tag{6}$$

with $P_k = \xi_k(X_{N,L,k}X_{N,L,k}^H)^{-1}$ , $\mu_k = (1+\xi_k)^{-1}$ and $\|v\|_P^2 = v^HPv$.
Consider the classical identification scheme as depicted in Fig.(2): one has

$$d_{L,k} = \eta_{L,k} - X_{N,L,k}W^{o\,H} \quad , \tag{7}$$

where $-W^o$ is the unknown FIR filter to identify and $\eta_{L,k} = [\ \eta_k\ \cdots\ \eta_{k-L+1}\ ]^H$ , $\eta_k$ being an *i.i.d.* centered sequence with finite variance.
One find straightforwardly the following recursion of the deviation filter $\widetilde{W}_{N,k} = W_{N,k} - W^o$

$$\widetilde{W}_{N,k} = \widetilde{W}_{N,k-1}\left(I - \mu_k X_{N,L,k}^H \left(X_{N,L,k}X_{N,L,k}^H\right)^{-1} X_{N,L,k}\right) - \mu_k \eta_{L,k}^H \left(X_{N,L,k}X_{N,L,k}^H\right)^{-1} X_{N,L,k} \quad . \tag{8}$$

At first glance, eq.(8) ressorts the projection process associated with the AP algorithm. In fact, when $\mu_k = 1$ , $\Phi_k^{AP} = I - X_{N,L,k}^H\left(X_{N,L,k}X_{N,L,k}^H\right)^{-1}X_{N,L,k}$ is the orthogonal projection matrix onto the orthogonal subspace to the subspace spanned by the columns of $X_{N,L,k}^H$. These columns are the $L$ most recent regressor vectors of the adaptive filter. In the case of the NLMS algorithm, the update of the filter deviation has the same form and the projection matrix is

$\Phi_k^{NLMS} = I - \frac{1}{\|X_N(k)\|^2} X_N(k) X_N(k)^H$ which is the projection onto the orthogonal subspace to $X_N(k)$.
The AP algorithm is known to converge faster than the NLMS algorithm, even when the input signal is white. This is explained from the projection scheme since applying $\Phi_k^{AP}$ to a given $\widetilde{W}_{N,k-1}$ gives a smaller norm vector than the one which would be obtained by applying $\Phi_k^{NLMS}$.
Another interesting property of the AP algorithm concerns the vector of a posteriori filtering errors which is

$$\begin{aligned} \epsilon_{N,L,k} &= d_{L,k} + X_{N,L,k} W_{N,k}^H = \begin{bmatrix} \epsilon_N^H(k|k) \\ \vdots \\ \epsilon_N^H(k-L+1|k) \end{bmatrix} \\ &= (1-\mu_k)\, \epsilon_{N,L,k}^p \ , \end{aligned} \tag{9}$$

as we can see, the vector of a posteriori filtering errors gets nulled if $\mu_k = 1$. This fact constitutes also a generalization to the NLMS algorithm where the a posteriori filtering error is nulled when $\mu_k = 1$.
The major drawback of the AP algorithm is its noise amplification which at first instance can be explained from eq.(8) where $\left(X_{N,L,k} X_{N,L,k}^H\right)^{-1}$ multiply $\eta_{L,k}$. So, when the inverse of the sample covariance matrix is large, the noise is amplified and disturbs the projection scheme. This is better seen by using the Singular Value Decomposition (SVD) of the data matrix $X_{N,L,k}$ : $X_{N,L,k} = U \, \Sigma \, V$ with $U$ a $L \times L$ matrix such that $UU^H = I$ , $\Sigma$ a $L \times L$ diagonal matrix where each diagonal element is called a singular value and $V$ is a $L \times N$ matrix such that $VV^H = I$. Using the SVD, eq.(8) becomes

$$\widetilde{W}_{N,k} = \widetilde{W}_{N,k-1} \Phi_k^{AP} - \mu_k \eta_{L,k}^H U \Sigma^{-H} V \ , \tag{10}$$

hence, noise amplification is related to the singular values of the data matrix $X_{N,L,k}$ and becomes more important as singular values decrease.
In [5], [11] and [13] modifications in the covariance matrix estimation scheme are done in order to reduce the noise amplification. [5] replaces the original sample covariance matrix $X_{N,L,k} X_{N,L,k}^H$ by $X_{N,L,k} X_{N,L,k}^H + \delta I$ ($\delta$ being a small positive number) a biased but regularized covariance matrix while [11] and [13] replaces the rectangular window of length $N$ over which the covariance matrix is estimated by an infinite exponentially weighted window.
The computational complexity of the AP algorithm is $2N + \mathcal{O}(L^2)$ when computing recursively the sample covariance inverse matrix. Fast versions of the AP algorithm have been derived in [5] and [11] by using judiciously the shift-invariance present in the algorithm due to the successive data-matrices $X_{N,L,i}$. It uses also the SWCFTF algorithm which is a fast version of the SWCRLS algorithm in order to compute recursively the sample covariance inverse matrix. This is briefly presented in the next section.

## 3 The FAP Algorithm

The FAP algorithm [5] in its relaxed form ($\mu_k \neq 1$) is given in Table I where $A_{L-1,k}$ and $B_{L-1,k}$ are respectively the forward and backward prediction filters of order $L-1$, $\alpha_{L-1}(k)$ and $\beta_{L-1}(k)$ are the corresponding prediction error variances (these quantities are computed

**Table I : Non-Relaxed FAP Algorithm**

| # | Computations | Cost per sample |
|---|---|---|
| 1 | $s_{L-1,k} = s_{L-1,k-1} - x_{k-N}X_{L-1}(k-N-1) + x_k X_{L-1}(k-1)$ | 2L |
| 2 | $\widehat{\epsilon}^p_N(k) = d_k + \widehat{W}_{N,k-1}X_N(k)$ | N |
| 3 | $\epsilon^p_N(k) = \widehat{\epsilon}^p_N(k) - \mu_k\,(E_{L,k-1})^H_{1:L-1}\,s_{L-1,k}$ | L |
| 4 | $\epsilon^p_{N,L,k} = \begin{bmatrix} \epsilon^p_N(k) \\ (1-\mu_{k-1})\epsilon^p_{N,L-1,k-1} \end{bmatrix}$ | L |
| 5 | SWCFTF (prediction part) of order $L-1$ : Update of $A_{L-1,k}, \alpha_{L-1}(k), B_{L-1,k}, \beta_{L-1}(k)$ | 10L |
| 6 | $e^p_{N,L,k} = \begin{bmatrix} 0 \\ (1-\mu_{k-1})\xi^p_{N,L-1,k-1} \end{bmatrix} + A^H_{L-1,k}\alpha^{-1}_{L-1}(k)A_{L-1,k}\epsilon^p_{N,L,k}$ | 3L |
| 7 | $\begin{bmatrix} \xi^p_{N,L-1,k} \\ 0 \end{bmatrix} = e^p_{N,L,k} - B^H_{L-1,k}\beta^{-1}_{L-1}(k)B_{L-1,k}\epsilon^p_{N,L,k}$ | 2L |
| 8 | $E_{L,k} = \begin{bmatrix} 0 \\ (E_{L,k-1})_{1:L-1} \end{bmatrix} + e^p_{N,L,k}$ | L |
| 9 | $\widehat{W}_{N,k} = \widehat{W}_{N,k-1} - \mu_k {E^L_{L,k}}^H X^H_N(k-L+1)$ | N |
| 10 | $W_{N,k} = \widehat{W}_{N,k} - \mu_k\,(E_{L,k})^H_{1:L-1}\,X_{N,L-1,k}$ | N(L−1) |
| Total cost per sample |  | 2N + 20L |

via the prediction part of an SWCFTF algorithm) and

$$E_{L,k} = \begin{bmatrix} e^{p\,1}_{N,L,k} \\ e^{p\,2}_{N,L,k} + e^{p\,1}_{N,L,k-1} \\ \vdots \\ e^{p\,L}_{N,L,k} + \cdots + e^{p\,1}_{N,L,k-L+1} \end{bmatrix} , \tag{11}$$

with

$$e^{p}_{N,L,k} = \begin{bmatrix} e^{p\,1}_{N,L,k} \\ \vdots \\ e^{p\,L}_{N,L,k} \end{bmatrix} = \left(X_{N,L,k} X^{H}_{N,L,k}\right)^{-1} \epsilon^{p}_{N,L,k} \ , \tag{12}$$

and finally

$$\xi^{p}_{N,L-1,k} = \left(X_{N,L-1,k} X^{H}_{N,L-1,k}\right)^{-1} \epsilon^{p}_{N,L-1,k} \ . \tag{13}$$

When there is no need in the computation of the filter estimate coeficients $W^{i}_{N,k}$ (hence, eq.(10) of Table I and eq.(7) of table II are not computed) the complexity of the relaxed FAP algorithm is $2N+20L$. Such situation where one does not need to update the filter coefficients arises in applications such as acoustic echo cancellation where one is mainly interested in the computation of the a priori output error signal.
The non-relaxed form of the FAP is obtained by setting $\mu_k = 1$, in this case simplification occurs that leads to the non-relaxed FAP algorithm which computational complexity is $2N + 14L$ when the filter estimate $W_{N,k}$ is not evaluated. Note that eq.(1) in Table I and II can be computed in $L$ operations instead of $2L$ operations if one stores the $N$ products $x_{k-N+i} X_{L-1}(k-N+i-1)\ ,\ i = 1, \ldots, N-1$. Note also that the update of the filter estimate $W_{N,k}$ needs $N(L-1)$ operations but the product $(E_{L,k})^{H}_{1:L-1} X_{N,L-1,k}$ corresponds to a convolution and can be computed by using fast convolution techniques that will reduce the amount of operations to $\mathcal{O}\left(N \log_2 L/L\right)$. This kind of computational reduction will be seen later.
In order to reduce the amount of computations in the FAP algorithm, we use the SUS. This idea comes from the fact that when one deals with relatively long adaptive filters, it is not necessary to update the filter at each new input sample because there is not a significant change in the filter coefficients after one update. The SUS does not necessarily involve approximations and the key ingredient is that even the adaptive filter is not updated all the time, it can be possible to compute efficiently the filtering errors that should be given if the filter estimate was updated at the sampling signal rate. The SUS leads to SU algorithms that are equivalent to the original algorithms, except for the fact that some quantities like the filter estimate are not provided at all the time instants. Moreover, fast convolution techniques can help to reduce the complexity of the SU algorithms and will give FSU algorithms. In the FAP algorithm, the prediction part SWCFTF algorithm is not computationally demanding and hence remains unchanged. This is also the case for all the computations which complexity is $\mathcal{O}(L)$. Henceforth, the objective is to reduce the $2N$ computational complexity that appears when updating the pseudo-filter $\widehat{W}_{N,k}$ and when computing the corresponding output error $\widehat{\epsilon}^{p}_{N}(k)$.

**Table II : Relaxed FAP Algorithm**

| # | Computations | Cost per sample |
|---|---|---|
| 1 | $s_{L-1,k} \;=\; s_{L-1,k-1} - x_{k-N} X_{L-1}(k-N-1) + x_k X_{L-1}(k-1)$ | 2L |
| 2 | $\widehat{\epsilon}^{\,p}_N(k) \;=\; d_k + \widehat{W}_{N,k-1} X_N(k)$ | N |
| 3 | $\epsilon^p_N(k) \;=\; \widehat{\epsilon}^{\,p}_N(k) - (E_{L,k-1})^H_{1:L-1}\, s_{L-1,k}$ | L |
| 4 | SWCFTF (prediction part) of order $L-1$ :<br>Update of $A_{L-1,k}, \alpha_{L-1}(k)$ | 10L |
| 5 | $E_{L,k} \;=\; \begin{bmatrix} 0 \\ (E_{L,k-1})_{1:L-1} \end{bmatrix} + \epsilon^p_N(k) A^H_{L-1,k} \alpha^{-1}_{L-1}(k)$ | L |
| 6 | $\widehat{W}_{N,k} \;=\; \widehat{W}_{N,k-1} - E^{L\,H}_{L,k} X^H_N(k-L+1)$ | N |
| 7 | $W_{N,k} \;=\; \widehat{W}_{N,k} - (E_{L,k})^H_{1:L-1}\, X_{N,L-1,k}$ | $N(L-1)$ |
| Total cost per sample | | 2N + 14L |

## 4 The SU FAP Algorithm

In what follows, we consider that the relaxation parameter is constant: $\mu_k = \mu$. We want to compute at time $k$ the pseudo-filter $\widehat{W}_{N,k}$ from its value at time $k-M$. Using the pseudo-filter update equation (see TableI or II), one finds straightforwardly

$$\begin{aligned} \widehat{W}_{N,k} &= \widehat{W}_{N,k-M} - \mu \mathcal{E}^H_{M,k} X_{N,M,k-L+1} \\ \mathcal{E}_{M,k} &= \left[ E^{L\,H}_{L,k} \cdots E^{L\,H}_{L,k-M+1} \right]^H \quad . \end{aligned} \tag{14}$$

Note that the computation of the different $E_{L,k-M+i}$ requires the computation of the $\widehat{\epsilon}^p_N(k-M+i)$ that are the successive output errors corresponding respectively to the successive pseudo-filters $\widehat{W}_{N,k-M+i}$ (see Table I or II). In fact, it turns out that the successive $\widehat{\epsilon}^p_N(k-M+i)$ can be computed without using the corresponding pseudo-filters $\widehat{W}_{N,k-M+i}$ . Consider for this the following vector of multistep a priori output errors

$$\widehat{\epsilon}_{N,M,k} = d_{M,k} + X_{N,M,k} W^H_{N,k-M} = \begin{bmatrix} \epsilon^H_N(k|k-M) \\ \vdots \\ \epsilon^H_N(k-M+1|k-M) \end{bmatrix} \quad . \tag{15}$$

Using (14), one can show that the one-step a priori output errors $\widehat{\epsilon}^p_N(k-M+i)$, for $i = 2, \ldots, M$ are obtained from the corresponding multistep a priori output errors as follows

$$\begin{aligned} \widehat{\epsilon}^p_N(k-M+i) &= \widehat{\epsilon}_N(k-M+i|k-M+i-1) \\ &= \widehat{\epsilon}_N(k-M+i|k-M) + \sum_{j=1}^{i-1} E^{L-1\,H}_{L,k-M+j} \tau_j(k-M+i) \quad , \end{aligned} \tag{16}$$

where

$$\tau_j(k-M+i) = X^H_N(k-M+i) X^H_N(k-M-N+j+1) \quad , \tag{17}$$

now as this was done in [1], one can compute efficiently the inner products $\tau_j(k-M+i)$ that appears in (16) as

for $i = 2, \ldots, M$

$$\tau_1(k-M+i) = \tau_1(k-2M+i) + \sum_{l=1}^{M} \left( x_{k-M+i-l} x_{k-M-N+2-l} - x_{k-2M+i-l-N} x_{k-2M-2N+l+3} \right)$$

for $j = 1, \ldots, i-1$

$$\tau_{j+1}(k-M+i+1) = \tau_j(k-M+i) + x_{k-M+i-1} x_{k-M-N+2+j} - x_{k-M+i-N+2} x_{k-M-2N+3+j} \quad . \tag{18}$$

The computational complexity associated with the procedure described in (18) is $3M^2$ per $M$ samples while (16) takes $0.5M^2$ operations per $M$ samples. Eqs.(14), (15), (16) and (18) with the other $\mathcal{O}(L)$ operations of the FAP algorithm give the SU FAP algorithm. Eqs.(14) and (15) are computationaly expensive but computational complexity of the new algorithm can be further reduced by using the FFT as is shown in the next section.

## 5 Fast computation using the FFT

It is possible to reduce the computational complexity of SU FAP by introducing FFT techniques as explained in [18]. In what follows, we shall assume for simplicity that $M$ is a power of two and that $K = N/M$ is an integer.
In order to get $\widehat{\epsilon}^{p}_{N,L,k}$ in (14), we need to compute the product $X_{N,M,k}\widehat{W}^{H}_{N,k-M}$. Consider the Hermitian transpose of this last product: it has the form $\upsilon_{N,k}\, X^{H}_{N,M,k}$ where $\upsilon_{N,k}$ is a row vector of $N$ elements.
Consider a partitioning of $\upsilon_{N,k}$ in $K$ subvectors of length $M$:

$$\upsilon_{N,k} = \left[\, \upsilon^{1}_{N,k} \, \cdots \, \upsilon^{K}_{N,k} \,\right] \quad , \tag{19}$$

and a partitioning of $X_{N,M,k}$ in $K$ submatrices of order $M \times M$:

$$X_{N,M,k} = [\, X_{M,M,k}\; X_{M,M,k-M} \;\cdots X_{M,M,k-N+M-1} \,] \quad , \tag{20}$$

then

$$\upsilon_{N,k}\, X^{H}_{N,M,k} = \sum_{j=1}^{K} \upsilon^{j}_{N,k} X^{H}_{M,M,k-(j-1)L} = \sum_{j=1}^{K} \left(\upsilon^{j}_{N,k} J\right)\left(J X^{H}_{M,M,k-(j-1)L}\right) \quad , \tag{21}$$

where $J$ is the reverse matrix

$$J = \begin{bmatrix} 0 & 0 & \cdots & 0 & 1 \\ 0 & 0 & \cdots & 1 & 0 \\ \vdots & & \iddots & & \vdots \\ \vdots & \iddots & & & \vdots \\ 1 & 0 & \cdots & 0 & 0 \end{bmatrix} \quad . \tag{22}$$

Here we use the reverse matrix in order to transform the Hankel data matrix into a Toeplitz one. This is just for notational convenience. Hence, we have essentially $K$ times the product of a vector of length $M$ with a $M \times M$ Toeplitz matrix. Such a product can be efficiently computed in basically two different ways [18]. One way is to use fast convolution algorithms which are interesting for moderate values of $M$. Another way is to use the Overlap-Save method. We can embed the $M \times M$ Toeplitz matrix $JX^{H}_{M,M,k}$ into a $2M \times 2M$ circulant matrix, viz.

$$\underline{X}^{H}_{M,M,k} = \begin{bmatrix} * & JX^{H}_{M,M,k} \\ JX^{H}_{M,M,k} & * \end{bmatrix} = \mathcal{C}\left(x^{H}_{2M,k} J\right) \quad , \tag{23}$$

where $\mathcal{C}(c^{H})$ is a right shift circulant matrix with $c^{H}$ as first row. Then we get for the vector-matrix product

$$\upsilon^{j}_{N,k} X^{H}_{M,M,k-(j-1)M} = \left[\, 0_{1\times M}\; \upsilon^{j}_{N,k} \,\right] \mathcal{C}\left(x^{H}_{2M,k-(j-1)MJ}\right) \begin{bmatrix} I_M \\ 0_{M\times M} \end{bmatrix} \quad . \tag{24}$$

Now consider the Discrete Fourier Transform (DFT) $\mathcal{V}^{j}_{N,k}$ of $\upsilon^{j}_{N,k}$

$$\mathcal{V}^{j}_{N,k} = \upsilon^{j}_{N,k}\, F_M \quad , \tag{25}$$

$F_M$ is the $M \times M$ DFT matrix whose generic element is $(F_M)_{p,q} = e^{-i2\pi\frac{(p-1)(q-1)}{M}}$, $i^2 = -1$. The inverse of $F_M$ is $\frac{1}{M}F_M^H$. It defines the inverse DFT transformation (IDFT)

$$v_{N,k}^j = \mathcal{V}_{N,k}^j \, \frac{1}{M} F_M^H \quad . \tag{26}$$

The product of a row vector $v$ with a circulant matrix $\mathcal{C}(c^H)$ where $v$ and $c$ are of length $m$ can be computed efficiently as follows. Using the property that a circulant matrix can be diagonalized via a similarity transformation with a DFT matrix, we get

$$v\,\mathcal{C}(c^H) \;=\; v\,F_m \;\operatorname{diag}\left(c^H F_m\right)\frac{1}{m}F_m^H \;=\; \left[(vF_m)\;\;\operatorname{diag}\left(c^H F_m\right)\right]\;\frac{1}{m}F_m^H \;, \tag{27}$$

where diag$(w)$ is a diagonal matrix with the elements of the vector $w$ as diagonal elements. So the computation of the vector in (24) requires the padding of $v$ with $M$ zeros, the DFT of the resulting vector, the DFT of $Jx_{2M,k-(j-1)M}$, the product of the two DFTs, and the (scaled) IDFT of this product. When the FFT is used to perform the DFTs, this leads to a computationally more efficient procedure than the straightforward matrix-vector product which would require $M^2$ multiplications. Note that at time $k$, only the FFT of $Jx_{2M,k}$ needs to be computed; the FFTs of $Jx_{2M,k-jM}, j = 1, \ldots, M-1$ have been computed at previous time instants.
This procedure is used for the product of $X_{N,M,k}$ with $W_{N,k-M}$ in (16). This reduces the $N$ computations per sample that are needed to

$$N\left[\frac{\mathrm{FFT}(2M)}{M^2} + \frac{2}{M}\right] + \frac{2\mathrm{FFT}(2M)}{M} \tag{28}$$

computations per sample (FFT$(M)$ signifies the computational complexity associated with a FFT of length $M$) or basically $\mathcal{O}\left(N\,\frac{\log_2(M)}{M}\right)$ operations.

## 6 The FSU FAP algorithm

Using the results of the previous section, we can reduce the computational complexity of the SU FAP algorithm by reducing the amount of operations needed for the computation of the two products in (14) and (15) that involve the two Hankel data matrices $X_{N,M,k-L+1}$ and $X_{N,M,k}$. For data storage considerations, it is better to use the data matrix $X_{N,M',k}$ , $M' = M+L-1$ which contains the two previous data matrices. By using $X_{N,M',k}$, we reduce the the amount of data storage because we just need to store $N/M'$ vectors of length $2M'$. This corresponds to a total of $2N$ data while $4N$ data need to be stored when considering the two data matrices $X_{N,M,k-L+1}$ and $X_{N,M,k}$. Moreover, it appears that the computational complexity for performing the matrix products involved are $\left(4 + 2\frac{N}{M}\right) FFT(2M) + 4N$ when using the two data matrices and $\left(3 + 2\frac{N}{M'}\right) FFT(2M') + 4N$ when using $X_{N,M',k}$. Both complexities are of the same order. Now, since we use the same data matrix, we have to compute the following vector of multistep output errors

$$\hat{\epsilon}_{N,M',k} = d_{M',k} + X_{N,M',k}W_{N,k-M}^H \tag{29}$$

and the vector of a priori errors at time $k$ which is used in the FAP algorithm is obtained by taking the first $M$ components of $\widehat{\epsilon}_{N,M',k}$

$$\widehat{\epsilon}^{\,p}_{N,M,k} = \left(\widehat{\epsilon}^{\,p}_{N,M',k}\right)_{1:M} \quad . \tag{30}$$

The vector of $M'$ a priori errors is computed as was explained in the previous section by sharing the product $X_{N,M',k}W^H_{N,k-M}$ into $N/M'$ subproducts that are computed via the FFT. Doing like this, the product (29) takes $\left(1+\frac{N}{M'}\right)\frac{FFT(2M')}{M}+2\frac{N}{M}$ operations per sample.
On the other hand, (14) is computed as follows

$$\widehat{W}_{N,k} = \widehat{W}_{N,k-M} - \mu \left[\, 0_{L-1}\ \mathcal{E}^H_{M,k} \,\right] X_{N,M',k} \ \ , \tag{31}$$

we can further reduce the $NM'$ multiplications per $M$ samples which are needed for the computation of $\left[\, 0_{L-1}\ \mathcal{E}^H_{M,k} \,\right] X_{N,M',k}$, by using the FFT technique. Consider the following decomposition of $X_{N,M',k}$ in $K$ submatrices of order $M' \times M'$

$$JX_{N,M',k} = \left[JX_{M',M',k} \ \ \cdots \ \ JX_{M',M',k-N+M_k}\ \right] \ \ , \tag{32}$$

and hence, the product in (31) becomes

$$\begin{aligned}&\left[\, 0_{L-1}\ \mathcal{E}^H_{M,k} \,\right] X_{N,M',k} \\ &= \ \left[ \left[ \left(\mathcal{E}^H_{M,k}J\right) \ \ 0_{L-1} \right] JX_{M',M',k} \ \cdots \ \left[ \left(\mathcal{E}^H_{M,k}J\right) \ \ 0_{L-1} \right] JX_{M',M',k-N+M'-1} \right] \ \ ,\end{aligned} \tag{33}$$

where every subproduct in (33) is computed as follows:

$$\begin{aligned}&\left[ \left(\mathcal{E}^H_{M,k}J\right) \ \ 0_{L-1} \right] JX_{M',M',k} = \left[ 0_{M'} \ \left(\mathcal{E}^H_{M,k}J\right) \ \ 0_{L-1} \right] \underline{X}^H_{M',M',k} \begin{bmatrix} I_{M'} \\ 0_{M'\times M'} \end{bmatrix} \\ &= \ \left[ \left( \left[ 0_{M'} \ \left(\mathcal{E}^H_{M,k}J\right) \ \ 0_{L-1} \right] F_{2L} \right) \mathrm{diag}(x^H_{2M',k}JF_{2M'}) \right] \frac{1}{2M'} F^H_{2M'} \begin{bmatrix} I_{M'} \\ 0_{M'\times M'} \end{bmatrix} \ \ ,\end{aligned} \tag{34}$$

As it is shown in (34), the product $\left[\, 0_{L-1}\ \mathcal{E}^H_{M,k} \,\right] X_{N,M',k}$ in (31) is done by adding $M'$ zeros to $\left[\, 0_{L-1}\ \mathcal{E}^H_{M,k} \,\right]$, computing the corresponding DFT, computing its product with the different DFTs of $Jx_{2M',k-jM'}$ , $j = 1, \ldots, M'-1$ , applying the IDFT to the different products and finally taking the first $M'$ elements of the results. Note that the different DFTs of $Jx_{2M',k-jM'}$ have been already computed at previous instants and that the DFT of $Jx_{2M',k}$ has been computed in (29). Hence, (31) is done in $\left(\frac{N}{M'}+1\right)\frac{FFT(2M')}{M}+2\frac{N}{M}$ multiplications per sample.
Now, in the case where one needs to compute the filter estimate coefficients, it is better for reducing the memory storage requirement to use the data matrix $X_{N,M',k}$ instead of $X_{N,L-1,k}$

$$W_{N,k} = \widehat{W}_{N,k} - \mu \left[\, (E_{L,k})^H_{1:L-1}\ 0_M \,\right] \ X_{N,M',k} \ \ , \tag{35}$$

Eq.(35) is done as eq.(31) in $\left(1+\frac{N}{M'}\right)\frac{FFT(2M')}{M}+2\frac{N}{M}$ operations per sample.
The resulting FSU FAP algorithm is summarized in Table III and IV, where we give respectively the non-relaxed and relaxed forms.

## 7 Concluding remarks

We have derived a new algorithm that is equivalent to the FAP algorithm. The computational complexity of the FSU FAP algorithm is $\mathcal{O}\left(\left(3+2\frac{N}{M'}\right)\frac{FFT(2M')}{M}+4\frac{N}{M}+3.5M+20L\right)$ operations per sample for the non-relaxed form and $\mathcal{O}\left(\left(3+2\frac{N}{M'}\right)\frac{FFT(2M')}{M}+4\frac{N}{M}+3.5M+14L\right)$ for the relaxed form. This can be very interesting for long filters. For example, when $(N,M,L)=(4096,256,16);(8192,256,16)$ and the FFT is done via the split radix ($FFT(2m)=mlog_2(2m)$ real multiplications for real signals) the multiplicative complexity is respectively $0.39N$ and $0.23N$ (we consider the non-relaxed form) compared to $7N$ for the FTF algorithm, the currently fastest RLS algorithm and $2N$ for the FNTF algorithm. Moreover, it seems that the $\mathcal{O}(L)$ computational complexity of the FSU FAP (and FAP) may be reduced by using a batch processing. This is the object of ongoing research. The cost we pay to achieve such complexity is a processing delay which is of the order of $M$ samples. The low computational complexity of the FSU FAP when dealing with long filters and its performance capabilities render it very interesting for applications such as acoustic echo cancellation. Nevertheless, the AP algorithm presents a drawback that is the noise amplification which originates from the short recangular window of length $N$ over which a $L\times L$ covariance matrix is estimated. Replacing the rectangular window by an exponentially weighted window will solve the problem of the noise amplification but unfortunately will attenuate the tracking ability of the algorithm. In order to reduce the noise amplification while keeping the tracking ability, we propose to replace the rectangular window by a generalized window which consists of two parts: first part is a rectangular window of lengh $L$ and second part is an exponentially weighted window that constitutes a tail to the first window and has smaller amplitude. Moreover, the amplitude of the exponential window can be time-varying. This idea seems to be very promising and can bring an answer to the classical tracking *vs.* noise amplification compromise in adaptive filtering.

**Table III : Non-Relaxed FSU FAP Algorithm**

| # | Computations | Cost per sample |
|---|---|---|
| 1 | $s_{L-1,k} \;=\; s_{L-1,k-1} - x_{k-N}X_{L-1}(k-N-1) + x_k X_{L-1}(k-1)$ | 2L |
| 2 | $\epsilon^p_{N,M,k} \;=\; \left(d_{M',k} + X_{N,M',k}\widehat{W}^H_{N,k-M}\right)_{1:M}$ | $\left(2+\frac{\mathrm{N}}{\mathrm{M}'}\right)\frac{\mathrm{FFT(2M')}}{\mathrm{M}} + 2\frac{\mathrm{N}}{\mathrm{M}}$ |
| 3 | Computation of the $\tau_j(k-M+i)\;\; i=1,\ldots,M\;\;,\;\; j=1,\ldots,i-1$ | 3M |
| 4 | $\widehat{\epsilon}^p_N(k-M+i) \;=\; \widehat{\epsilon}_N(k-M+i\|k-M) + \sum_{j=1}^{i-1} E^{L\;H}_{L,k-M+j}\tau_j(k-M+i)$ | .5M |
| 5 | $\epsilon^p_N(k) \;=\; \widehat{\epsilon}^p_N(k) - \mu\,(E_{L,k-1})^H_{1:L-1}\,s_{L-1,k}$ | L |
| 6 | $\epsilon^p_{N,L,k} \;=\; \begin{bmatrix}\epsilon^p_N(k)\\ (1-\mu)\epsilon^p_{N,L-1,k-1}\end{bmatrix}$ | L |
| 7 | SWCFTF (prediction part) of order $L$ :<br>Update of: $A_{L-1,k}, \alpha_{L-1}(k), B_{L-1,k}, \beta_{L-1}(k)$ | 10L |
| 8 | $e^p_{N,L,k} \;=\; \begin{bmatrix}0\\ (1-\mu)\xi^p_{N,L-1,k-1}\end{bmatrix} + A^H_{L-1,k}\alpha^{-1}_{L-1}(k)A_{L-1,k}\epsilon^p_{N,L,k}$ | 3L |
| 9 | $\begin{bmatrix}\xi^p_{N,L-1,k}\\ 0\end{bmatrix} \;=\; e^p_{N,L,k} - B^H_{L-1,k}\beta^{-1}_{L-1}(k)B_{L-1,k}\epsilon^p_{N,L,k}$ | 2L |
| 10 | $E_{L,k} \;=\; \begin{bmatrix}0\\ (E_{L,k-1})_{1:L-1}\end{bmatrix} + e^p_{N,L,k}$ | L |
| 11 | $\widehat{W}_{N,k} \;=\; \widehat{W}_{N,k-M} - \mu\left[\,0_{L-1}\;\; \mathcal{E}^H_{M,k}\,\right]X_{N,M',k}$ | $\left(1+\frac{\mathrm{N}}{\mathrm{M}'}\right)\frac{\mathrm{FFT(2M')}}{\mathrm{M}} + 2\frac{\mathrm{N}}{\mathrm{M}}$ |
| 12 | $W_{N,k} \;=\; \widehat{W}_{N,k} - \mu\left[\,(E_{L,k})^H_{1:L-1}\;\; 0_M\,\right]\;X_{N,M',k}$ | $\left(1+\frac{\mathrm{N}}{\mathrm{M}'}\right)\frac{\mathrm{FFT(2M')}}{\mathrm{M}} + 2\frac{\mathrm{N}}{\mathrm{M}}$ |
| Total cost per sample | | $\left(3+2\frac{\mathrm{N}}{\mathrm{M}'}\right)\frac{\mathrm{FFT(2M')}}{\mathrm{M}} + 4\frac{\mathrm{N}}{\mathrm{M}} + 3.5\mathrm{M} + 20\mathrm{L}$ |

**Table IV : Relaxed FSU FAP Algorithm**

| # | Computations | Cost per sample |
|---|---|---|
| 1 | $s_{L-1,k} \;=\; s_{L-1,k-1} - x_{k-N} X_{L-1}(k-N-1) + x_k X_{L-1}(k-1)$ | 2L |
| 2 | $\epsilon^p_{N,M,k} \;=\; \left(d_{M',k} + X_{N,M',k} \widehat{W}^H_{N,k-M}\right)_{1:M}$ | $\left(2+\frac{\mathrm{N}}{\mathrm{M}'}\right)\frac{\mathrm{FFT(2M')}}{\mathrm{M}} + 2\frac{\mathrm{N}}{\mathrm{M}}$ |
| 3 | Computation of the $\tau_j(k-M+i)$ $i=1,\ldots,M$ , $j=1,\ldots,i-1$ | 3M |
| 4 | $\widehat{\epsilon}^p_N(k-M+i) \;=\; \widehat{\epsilon}_N(k-M+i\|k-M) + \sum_{j=1}^{i-1} E^{L\,H}_{L,k-M+j}\tau_j(k-M+i)$ | .5M |
| 5 | $\epsilon^p_N(k) \;=\; \widehat{\epsilon}^p_N(k) - (E_{L,k-1})^H_{1:L-1}\, s_{L-1,k}$ | L |
| 6 | SWCFTF (prediction part) of order $L-1$ :<br>Update of: $A_{L-1,k}, \alpha_{L-1}(k)$ | 10L |
| 7 | $E_{L,k} \;=\; \begin{bmatrix} 0 \\ (E_{L,k-1})_{1:L-1} \end{bmatrix} + \epsilon^p_N(k) A^H_{L-1,k}\alpha^{-1}_{L-1}(k)$ | L |
| 8 | $\widehat{W}_{N,k} \;=\; \widehat{W}_{N,k-M} - \left[\, 0_{L-1}\; \mathcal{E}^H_{M,k} \,\right] X_{N,M',k}$ | $\left(1+\frac{\mathrm{N}}{\mathrm{M}'}\right)\frac{\mathrm{FFT(2M')}}{\mathrm{M}} + 2\frac{\mathrm{N}}{\mathrm{M}}$ |
| 9 | $W_{N,k} \;=\; \widehat{W}_{N,k} - \mu \left[\, (E_{L,k})^H_{1:L-1}\; 0_M \,\right] X_{N,M',k}$ | $\left(1+\frac{\mathrm{N}}{\mathrm{M}'}\right)\frac{\mathrm{FFT(2M')}}{\mathrm{M}} + 2\frac{\mathrm{N}}{\mathrm{M}}$ |
| Total cost per sample | | $\left(3+2\frac{\mathrm{N}}{\mathrm{M}'}\right)\frac{\mathrm{FFT(2M')}}{\mathrm{M}} + 4\frac{\mathrm{N}}{\mathrm{M}} + 3.5\mathrm{M} + 14\mathrm{L}$ |

## References

[1] J. Benesty, P. Duhamel, "A Fast Exact LMS Adaptive Algorithm". *IEEE Trans. on ASSP*, ASSP-29(3):2904–2920, Dec. 1992.

[2] J.M. Cioffi and T. Kailath, "Fast, recursive least squares transversal filters for adaptive filtering". *IEEE Trans. on ASSP*, ASSP-32(2):304–337, April 1984.

[3] J.M. Cioffi and T. Kailath, "Windowed Fast Transversal Filters Adaptive Algorithms with Normalization". *IEEE Trans. on ASSP*, ASSP-33(3):607–625, June 1985.

[4] E. Eleftheriou and D. Falconer, "Tracking properties and steady state performance of RLS adaptive filter algorithms". *IEEE Trans. on ASSP*, ASSP-34(5):821–823, July 1987.

[5] S. L. Gay, "A Fast Converging, Low Complexity Adaptive Filtering Algorithm". *int. rep. AT&T*, 1993.

[6] E. Hänsler, "The hands-free telephone problem. An annotated bibliography" *Signal Processing*, Vol. 27, pp. 259–271.

[7] S. Haykin, "Adaptive Filter Theory", Second edition, Prentice Hall, Englewood Cliffs, NJ, 1991.

[8] T. Kailath, *Linear Systems.* Prentice-Hall, Englewood Cliffs, NJ, 1980.

[9] T. Kailath, S.Y. Kung, and M. Morf, "Displacement ranks of matrices and linear equations". *J. Math. Anal. Appl.*, 68(2):295–407, 1979. (See also *Bull. Amer. Math. Soc.*, vol. 1, pp. 769–773, 1979.).

[10] K. Maouche and D. T. M. Slock, "The Fast Subsampled-Updating Fast Newton Transversal filter (FSU FNTF) Algorithm for Adaptive Filtering Based on on a Schur procedure and the FFT". Research Report N° 94-014, Institut Eurécom.

[11] M. Montazeri and P. Duhamel, "Fast Modified Projection Algorithms for Acoustic Echo Cancellation ". Submitted to *IEEE Trans. on ASSP*, 1994.

[12] G.V. Moustakides and S. Theodoridis, "Fast newton transversal filter –A new class of adaptive estimation algorithms". *IEEE Trans. on ASSP*, ASSP-39(10):2184–2193, Oct. 1991.

[13] C. B. Papadias and D. T. M. Slock, "New Adaptive Blind Equalization Algorithms for Constant Modulus Constellations". *International Conference on Acoustics, Speech and Signal Processing (ICASSP-94)*, pp. 321-324, Adelaide, Australia, April 19-22, 1994.

[14] T. Pétillon, A. Gilloire and S. Thedoridis, "The Fast Newton Transversal filter: An Efficient Scheme for acoustic Echo Cancellation in Mobile Radio ". *IEEE Trans. on ASSP*, ASSP-42(3):2184–2193, March 1994.

[15] D.T.M. Slock and T. Kailath, "Numerically Stable Fast Transversal Filters for Recursive Least-Squares Adaptive Filtering". *IEEE Trans. Signal Proc.*, ASSP-39(1):92–114, Jan. 1991.

[16] D.T.M. Slock and K. Maouche, “The Fast Subsampled-Updating Recursive Least-Squares (FSU RLS) Algorithm for Adaptive Filtering Based on Displacement Structure and the FFT”. *Signal Processing*, Vol. 40, No. 1, Oct. 1994, pp. 5–20.

[17] D.T.M. Slock and K. Maouche, “The Fast Subsampled-Updating Stabilized Fast Transversal Filter (FSU SFTF) RLS Algorithm”. In *Proc. EUSIPCO 94, VIIth European Signal Processing Conference,* pages 740-743, Edinburgh, Scotland, U.K. Sep. 13-16 1992.

[18] M. Vetterli, “Fast Algorithms for Signal Processing”. In M. Kunt, editor, *Techniques modernes de traitement numérique des signaux.* Presses Polytechniques et Universitaires Romandes, Lausanne, Switzerland, 1991. ISBN 2-88074-207-2.

[19] B. Widrow et al., “Stationary and nonstationary learning characteristics of the LMS adaptive filter”, *Proc. IEEE*, vol. 64, No. 8, August 1976, pp. 1151–1162.