Fast Transversal Recursive least-squares (FT-RLS) Algorithm

Dan J. Dechene
Department of Electrical and Computer Engineering
University of Western Ontario
London, Ontario, N6A 5B9, Canada
Email: ddechene@uwo.ca

***Abstract*—In this paper a brief overview of the Fast Transversal Recursive Least-Squares (FT-RLS) algorithm is provided. This algorithm is designed to provide similar performance to the standard RLS algorithm while reducing the computation order. This is accomplished by a combination of four transversal filters used in unison. Finite precision effects are also briefly discussed. Simulations are performed with both algorithms to compare both the computational burden as well as the performance of both schemes for adaptive noise cancellation. Simulations show that FT-RLS offers comparable performance with respect to the standard RLS in addition to a large reduction in computation time for higher order filters.**

***Index Terms*—Adaptive Filters, RLS, least-squares**

I. Introduction

Adaptive noise cancelation is being used as a prominent solution in a wide range of fields. From the standpoint of performance, it is widely known [1] that the Recursive Least-Squares (RLS) algorithm offers fast convergence and good error performance in the presence of both white and coloured noise. This robustness makes this algorithm highly useful for adaptive noise cancelation. Unfortunately even as the computational power of devices today increases, it remains largely difficult to utilize the RLS algorithm for real-time signal processing.

This is largely due to the computational complexity of the RLS algorithm. The computation time of the algorithm scales with $O(M^2)$, where M is the filter order. The result of which makes the computation of RLS in real-time nearly impossible (especially for larger filter lengths), even with the increasing computation power of equipment today.

In this paper we examine the *Fast Transversal RLS* (FT-RLS) filter. This filter is designed provide the least squares solution to the adaptive filtering problem in a manner that scales with $O(M)$, which makes it a more viable candidate for real-time applications.

The rest of this paper is organized in the following manner. Section II will provide an overview of the FT-RLS algorithm. In Section III several problems when implementing in finite precision will be discussed. Section IV will provide some simulation results of FT-RLS versus RLS and finally Section V will draw conclusions on this work.

II. Fast-Transversal Recursive least-squares

The Fast Transversal RLS (FT-RLS) filter is designed to provide the solution to the filtering problem with performance equal to the standard recursive least-squares (RLS) algorithm. In addition, the FT-RLS filter provides this solution with reduced computational burden which scales linearly with the filter order. This makes it a very attractive solution in real-time noise cancellation applications. The FT-RLS development is based on the derivation of lattice-based least-squares filters but has the structure of four transversal filters working together to compute update quantities reducing the computational complexity [2]. The full derivation of the FT-RLS algorithm can be found in [3]. The four transversal filters used for forming the update equations are:

1) *Forward Prediction:* The forward prediction transversal filter computes the forward filter weights in such a manner that minimizes the prediction error (in the least-squares sense) of the next input sample based on the previous input samples. This filter also computes the prediction error of estimation using both a priori and a posterior filter weights, in addition to the minimum weighted least squares error for this forward prediction.
2) *Backward Prediction:* The backward prediction transversal filter computes backward filter weights in such a manner that minimizes the prediction error (in the least-squares sense) of the $u(n-M)$ sample using the vector input $\mathbf{u}_b(n) = [u(n)u(n-1)\cdots u(n-M+1)]^T$. This filter will also compute the respective prediction error of estimation using both a priori and a posterior filter weights, in addition to the minimum weighted least-squares error for this backward prediction.
3) *Conversion Factor:* The gain computation transversal filter is used to recursively compute a gain vector which is used to updating the forward, backward and joint-process estimation filter weights. As such this filter also provides recursive computation of the factors relating a priori and a posteriori error quantities.
4) *Joint-Process Estimation:* The joint-process estimation transversal filter computes filter weights in such a manner that the error between the estimated signal and the **desired** input signal ($d(n)$) is minimized. It is the joint-process estimation weights that are equivalent to filter weights in other adaptive filtering algorithms.

The structural diagram of the FT-RLS using the sub-component transversal filters is shown in Figure 2 and the notation used is shown in Table I. For space limitation, the interested reader is referred to [1]–[3] for the full details of the algorithm design.

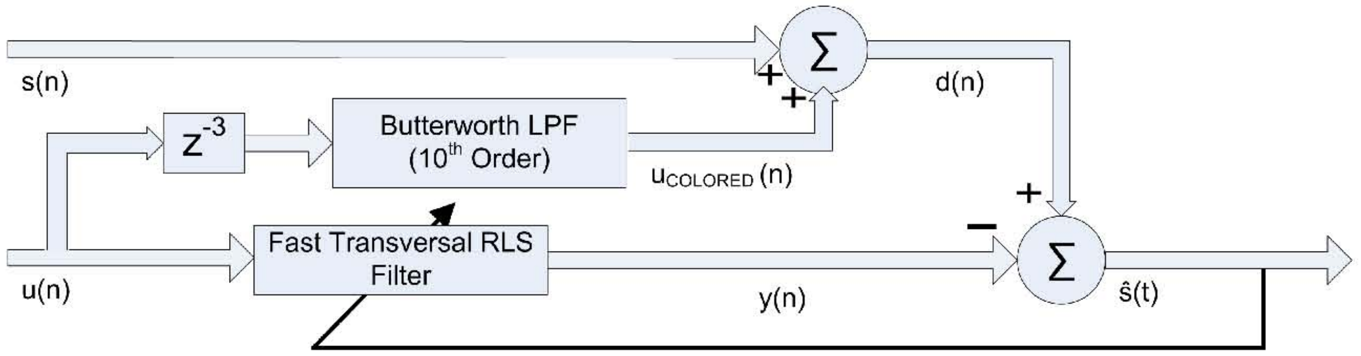


Fig. 1: Noise Cancellation System Example

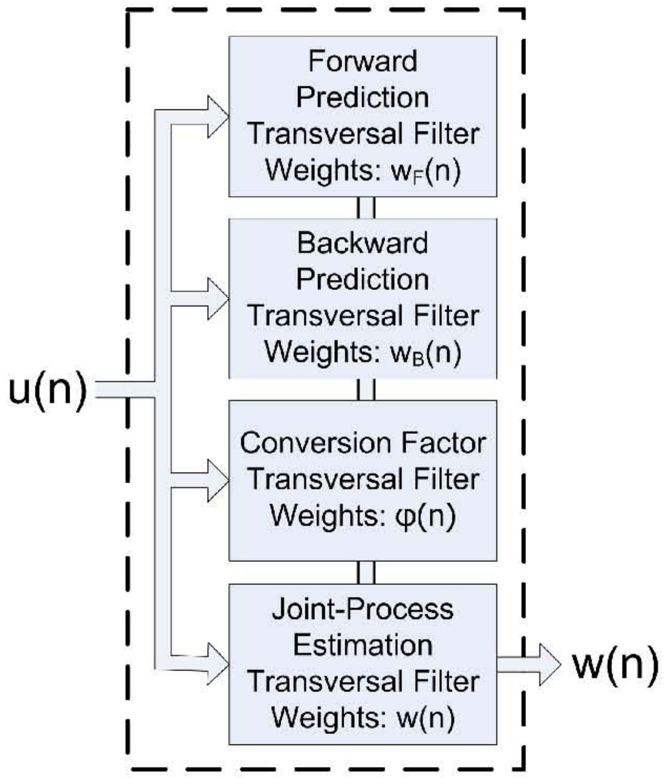


Fig. 2: Fast Transversal RLS Block Diagram

III. Finite Precision Implementation

The FT-RLS algorithm unfortunately leads to numerical instability in a finite precision environment. The details of this instability are studied in detail in [4]. There have been several methods proposed to create a stabilized version of FT-RLS. In general, these methods involve expressing certain update equations in different forms. With high precision these quantities are identical, however become unequal in finite precision. The difference between the values is often used as a measure of the numerical sensitivity. One solution is to utilize a weighted average of the update equations. This is the solution proposed in [4]. Using calculation redundancy in several of theses quantities provides numerical stability in the presence of finite precision. The result of this gives rise to the proposed stabilized FT-RLS [4], which requires more calculations per iteration to operate, however is more numerical robust compared to FT-RLS and maintains order scaling of $O(M)$.

IV. Simulation Results

In order to compare the performance of FT-RLS versus the standard RLS, both algorithms were implemented in MATLAB in an adaptive noise cancellation application. Simulations were performed using the system diagram shown in Figure 1. The input signal ($s(n)$) is a random audio signal and the reference noise ($u(n)$) is a white noise sequence. Simulation results are presented for the case where the SNR is fixed to 0dB. The number of algorithm iterations is set of 1000. The code used for the simulation of the FT-RLS algorithm can be found in the appendix.

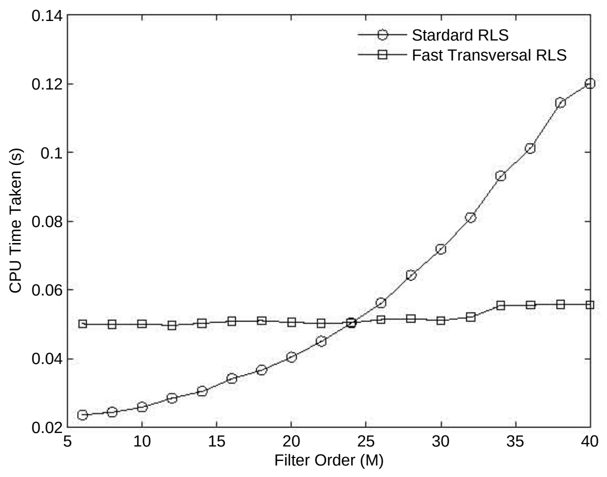
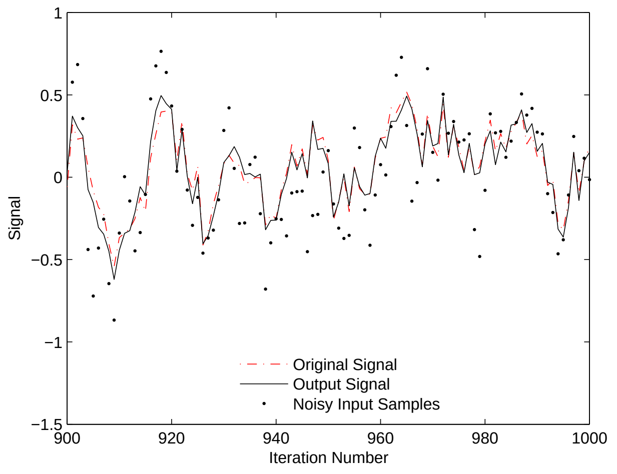


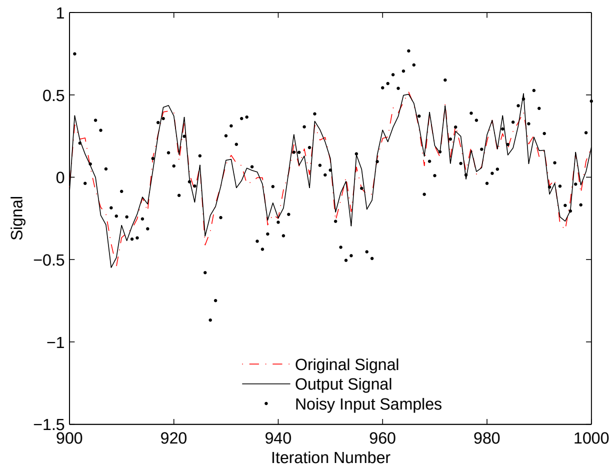
Fig. 3: Computation Time vs. Filter Order

A. Computational Burden

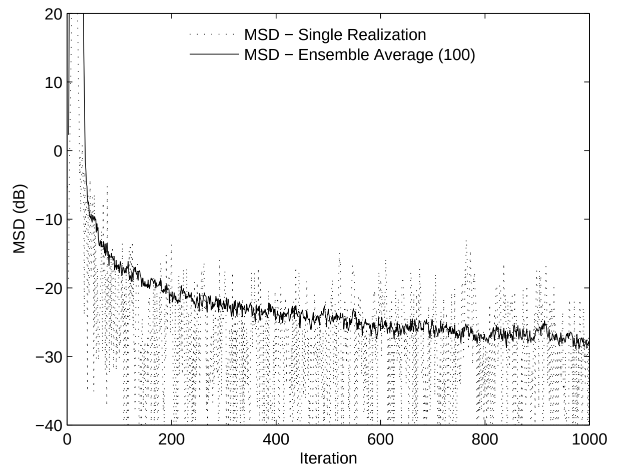
In order to first understand the improvements offered by the FT-RLS algorithm, the computational burdens with respect to the filter order must be found. For this, the example from Figure 1 is used, while varying the filter length for both algorithms. The time taken to process 1000 samples is averaged over 500 simulations and found for both algorithms. From Figure 3 it can be seen that the FT-RLS algorithm requires more computation time for small filter lengths. This



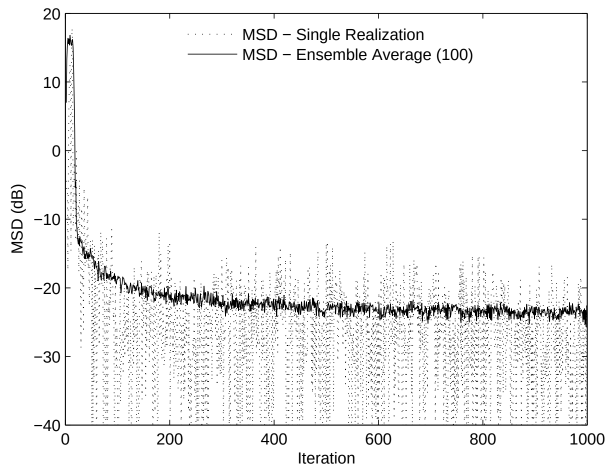
(a) Fast Transversal RLS - Signal



(b) Standard RLS - Signal



(c) Fast Transversal RLS - MSD



(d) Standard RLS - MSD

Fig. 4: Noise Cancellation with Filter Order of 25

is due in part to the larger number of operations required to be performed at each iteration, as the number of operations is irrespective of the filter order, the standard RLS algorithms tends to perform faster for short filters. At approximately a length of 25, the computation time suffered by both algorithms is equal. For lengths greater than this, it can be seen that the standard RLS algorithm requires exponentially more computation time. This is due to the large size of matrices involved in the algorithm computation.

It is important to note that the true computational performance is inhibited by utilizing MATLAB (versus using a fixed-point DSP) as the simulation environment. This is a result of MATLAB optimizing certain computations and therefore is a poor measurement of the actual computation time required for a hardware implementation. However from Figure 3 it is clear that the use of FT-RLS is advantageous.

B. Performance Comparison

In order to determine the performance of the algorithms for adaptive noise cancellation, the filter order is fixed to 25 (where both algorithms achieve similar computational time) and measurements are taken. Simulations are performed to determine how well the algorithms perform in noise cancellation. The waveforms containing the original, noisy and output signals for both algorithms are shown in Figures 4a and 4b. It can be seen that both algorithms perform well at an SNR of 0dB. The actual performance of FT-RLS under simulation actually performs slightly better than the standard RLS. Overall, both algorithms offer reasonable performance under the presence of low SNR. The minimum squared deviation (MSD) provides the actual deviation performance for these simulations and can be seen in Figures 4c and 4d respectively. The results for MSD are shown for both a single realization and the ensemble average over 100 simulations. It can be seen that both algorithms converge quickly and achieve similar MSD performance, however the FT-RLS MSD is slightly lower than the standard RLS. This is a result of the noise contained in the filtered output (shown in Figure 4b).

C. *Weight Convergence*

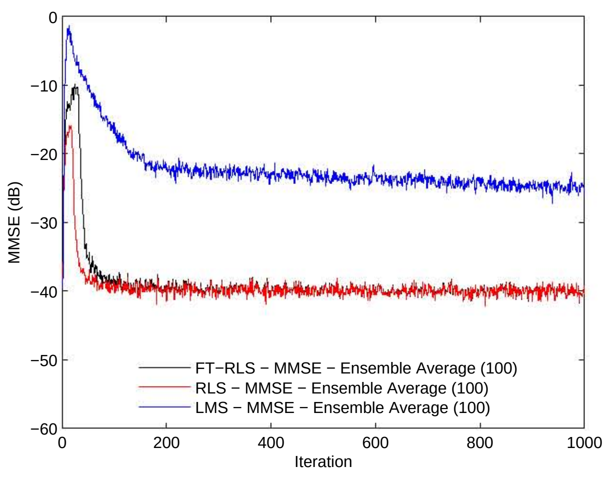


Fig. 5: Error Performance for System Identification

This paper also briefly compares the convergence rate and error floor of FT-RLS with the standard RLS and LMS algorithms. The input noise is a coloured noise sequence and the impulse response of the system being identified is 5 taps and generated at random. All filter lengths are set to 25 and the variance of the measurement noise is 0.001. It can be seen from Figure 5 that both the RLS and FT-RLS algorithms obtain nearly identical performance (fast convergence and low error floor) while LMS is hindered in performance. These results confirm that the FT-RLS is comparable to the standard RLS in terms of performance.

Quantity	Notation
Desired Input Samples (Noise + Signal)	$d(i-k)$
Filter Input Samples (White Noise)	$u(i-k)$
Filter Order	M
Filter Weights (Joint-Process)	$\mathbf{W}(i)$
Signal	$s(i)$
Forward Prediction Filter Weights	$\mathbf{W}_f(i)$
Backward Prediction Filter Weights	$\mathbf{W}_b(i)$
Conversion Factor Filter Weights	$\phi(i)$

TABLE I: Notation

V. Conclusion

In this paper, an overview of the FT-RLS algorithm was provided. After thorough simulations, it is clear that the FT-RLS algorithm is a highly suitable solution for adaptive filtering applications where a large filter order is required without sacrificing the performance offered by the standard RLS algorithm in the presence of both white and coloured noise. Future work should examine the feasibility of a real-time hardware implementation of the FT-RLS algorithm.

Appendix
FT-RLS Matlab Code For Noise Cancellation

```
%NOISE GENERATION
M=25;   N=1000;   Delay=3;   lambda=1;
noise=randn(N,1);
fnoise=filter([zeros(1,Delay) 1],1,noise);
[Num,Den]=butter(10,0.5);
fnoise=filter(Num,Den,foise);
fnoise=fnoise/std(fnoise)*0.5;
weights=zeros(M,length(fnoise)+1);
d=signal+fnoise;
%INITIALIZATION
epsilon=0.00001;
w=zeros(M,N);
uvec=zeros(1,M);
e=zeros(N,1);
wf(1:M,1)=0;
wb(1:M,1)=0;
w(1:M,1)=0;
phi(1:M,1)=0;
gamma(1)=1;
epb(1:M,1)=epsilon;
epf(1:M,1)=epsilon;

%FILTERING ALGORITHM
for i=2:length(fnoise)
   %Forward A Priori Prediction Error
     efa(i)=noise(i)-uvec*wf(:,i-1);
   %Forward A Posterior Prediction Error
     ef(i)=efa(i)*gamma(i-1);
   %MWLS Forward Error
     epf(i)=lambda*epf(i-1)+efa(i)*(ef(i))';
   %Forward Weight Update
     wf(:,i)=wf(:,i-1)+phi(:,i-1)*ef(i);
     phi1(:,i)=[0;phi(:,i-1)]+efa(i)'/(lambda*epf(i-1))*[1;-wf(:,i-1)];
   %M+1 Conversion Factor
     gamma1(i)=gamma(i-1)*lambda*epf(i-1)/epf(i);
   %Backward A Priori Prediction Error
     eba(i)=lambda*epb(i-1)*phi1(M+1,i);
   %M Conversion Factor
     gammainv=1/gamma1(i)-phi1(M+1,i)*eba(i);
     gamma(i)=1/gammainv;
   %Backward A Posterior Prediction Error
     eb(i)=eba(i)*gamma(i);
   %MWLS Backward Error
     epb(i)=lambda*epb(i-1)+eb(i)*eba(i)';
   %M Conversion Weight
     newvec=phi1(:,i)-phi1(M+1,i)*[-wb(:,i-1);1];
     phi(:,i)=newvec(1:M);
   %Backward Weight Update
     wb(:,i)=wb(:,i-1)+phi(:,i)*eb(i);
   %Update with New Sample of Input Data
     uvec=[noise(i) uvec(1:M-1)];
   %A Priori Joint-Estimation Error
     ea(i)=d(i)-uvec*w(:,i-1);
   %A Posterior Joint-Estimation Error
     e(i)=ea(i)*gamma(i);
   %Joint-Estimation Weight Update
     w(:,i)=w(:,i-1)+e(i)*phi(:,i);
end
```

References

[1] S. Haykin, *Adaptive Filter Theory*, 4th ed. Pearson Education, 2002.

[2] P. S. R. Diniz, *Adaptive Filtering: Algorithms and Practical Implementation*. Kluwer Academic Publishers, 1997.

[3] J. Cioffi and T. Kailath, “Fast Recursive-Least-Squares, Transversal Filters for Adaptive Filtering,” *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 32, pp. 304–337, 1984.

[4] D. T. M. Slock and T. Kailath, “Numerically Stable Fast Transversal Filters for Recursive Least Squares Adaptive Filtering,” *IEEE Trans. on Signal Processing*, vol. 39, pp. 92–113, Jan 1991.