Efficient Signal Processing Algorithms for Passive Radars

Michał Meller
Department of Signal and Information Processing,
Bumar Elektronika, Gdańsk Division, ul. Hallera 233A, 80-502 Gdańsk
POLAND

michal.meller@bumar.com

ABSTRACT

Due to their reliance on transmitters of opportunity, passive radars require special signal processing algorithms. Adaptive filtering is used to filter out strong stationary echos, while computation of crossambiguity of reference and echo signals is necessary to perform target detection. At present stage, signal processing for FM radars can be implemented on a standard PC without much effort. However, the task becomes more challenging for wideband signals, such as DVB-T.

The paper surveys several algorithms for adaptive filtering and computation of crossambiguity function. Adaptive filtering covers: block algorithms (least squares estimator and derivatives), least mean square (LMS) algorithms (standard LMS, Block LMS, Fast LMS, Delay-LMS), lattice estimators (LS lattice filter, gradient adaptive lattice). The concept of frequency response of adaptive filtering algorithms is introduced and an appropriate analysis is performed.

The second part of the paper focuses on fast computation of crossambiguity via the fast Fourier transform (FFT) algorithm. Again, the issue of frequency response of the resulting implementation is discussed.

1.0 INTRODUCTION

Passive Coherent Location (PCL), often referred to as passive radar, is a prospective technique of detecting and tracking objects. Unlike classical radars, passive radars do not employ dedicated transmitters and fully rely on third party ones, so-called illuminators of opportunity. Typical sources of illumination include, but are not limited to: radio and television broadcasting stations – both analog [1] and digital [2] (digital audio broadcasting DAB, digital video broadcasting DVB), GSM signals[3], Wi-Fi/WiMAX networks [4]. Among important advantages of PCL, as compared to the classical approach, one could point to the following ones

- Better utilization of available spectrum. Passive radar sensors do not occupy RF spectrum, which makes them well suited for application in congested environment.
- Increased difficulty to detect and localize the radar. Passive radar remains silent during its operation, which greatly improves its chance of survival.
- Improved jamming resistance of PCL. Random characteristics of typical sources of illumination combined with coherent signal processing techniques improve jamming robustness of passive radars. Interestingly, passive radar can (at least theoretically) take advantage of jammer's presence – by using the jammer as a source of illumination.

- Possibility to detect low-observable targets. `Stealth' is achieved by a combination of careful shaping and application of radar absorbing materials. Even though material science advanced considerably over the last years, it is believed that shape remains the primary factor in achieving low radar cross section. Loosely speaking, the aim of shaping is to reflect the incoming waves away of its source in a controlled fashion. This technique works very well against classical radars with co-located transmitter and receiver, but has less chances of success against bistatic and multistatic radars, such as PCL.

Unfortunately, passive radars require a considerably more sophisticated signal processing than their classical counterparts. Typical passive radar signal processing subsystem includes the following elements (Figure 1):

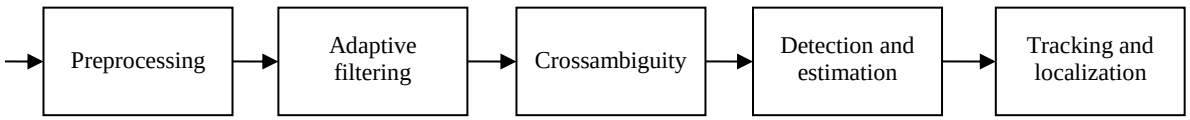

Figure 1: Block diagram of signal processing in a typical passive radar.

- Preprocessing. The goal of this stage is to signal quality prior to the later stages of processing. Preprocessing usually includes some form of preliminary filtering, IQ downconversion, decimation and digital beamforming.
- Adaptive filtering. One of the greatest challenges in PCL is dynamic range. Useful echoes are usually very weak and masked by strong direct signal and clutter. The difference in dynamics of targets and unwanted components of the signal is usually very large, possibly exceeding 60 dB. This makes target detection very difficult. Adaptive filtering allows one to improve dynamic range of the radar by removing (subtracting) the unwanted strong signal. Such an operation is possible thanks to the fact that unwanted signal components are stationary or slowly time-varying, while targets usually introduce a considerable Doppler shift to the echo. This makes it possible for the adaptive filter to "distinguish" between wanted and unwanted components of the signal [5].
- Computation of crossambiguity. Crossambiguity is a PCL equivalent of matched filtering and Doppler processing. The operation is defined as

$$\chi(\tau,\omega)=\sum_{t=0}^{T-1} y'(t)x^{*}(t-\tau)e^{-j\omega t}\ , \quad (1)$$

- where t denotes discrete time, τ and ω are the delay and the Doppler shift, respectively, $y'(t)$ is the echo signal (after adaptive filtering), $x(t)$ is the reference signal, z^{*} denotes the complex conjugate of z and T is the integration time. The values of $\chi(\tau,\omega)$ are computed on a grid of some meaningful values of delay and Doppler.
- Detection and estimation. Once (1) is computed, target detection (possibly employing CFAR) can be performed. Afterwards, the bistatic range, velocity and (optionally) azimuth/elevation of the detected targets are estimated.
- Tracking and localization. Quality of the preliminary estimates of bistatic range and velocity may be improved using tracking [6], [7]. The position of the target in Cartesian coordinates can be estimated by cross-referencing information yielded by several base stations and/or several passive radars [8].

This paper focuses on the second and the third step of processing chain. Its primary aim is to survey a range of techniques which can be applied to perform adaptive filtering and computation of crossambiguity. We also evaluate their potential for application in passive radar. The reasons behind such a choice are outlined below

- Detection, estimation, tracking and localization operate on a substantially smaller datastreams than adaptive filtering and computation of crossambiguity, i.e. their efficiency is less critical.
- For narrowband signals, such as the FM broadcast (approx. 200 kHz), both the adaptive filtering and the computation of crossambiguity can be performed in real time using a modern standard PC. Unfortunately, requirements of both operations may grow faster than linear with sampling frequency. Linear growth of required processing speed is caused by smaller sampling period, i.e. less time to complete all computations. However, because of increased sampling rate it occurs that the orders of adaptive filters and the size of the grid (τ, ω) must be increased as well! This means that, while one can process FM signal with relative ease, PCL for broadband signals, such as DAB or DVB, is considerably more difficult.

The paper is organized as follows. Section 2 revises the concepts of computational complexity in the context of modern hardware computing platforms. Section 3 presents various adaptive filtering algorithms and discusses their strong points and weaknesses. Additionally, tracking behaviour of several adaptive filters is analyzed. Such an analysis allows one to determine the "bandwidth" of adaptation laws and to predict the width of the Doppler notch caused by adaptive filtering. Section 4 is devoted to computation of crossambiguity function. Section 5 concludes.

2.0 COMPUTATIONAL COMPLEXITY REVISED

It is common to express computational complexity using number of multiplications. Unfortunately, in the context of modern hardware, such an approach has serious flaws. With multiple cores and execution units running in parallel, multi-level cache memories, and pipelining one should also take into account the following factors

- Parallelism. It is desirable that every step of the algorithm consists of many simple, similar operations which can be executed independently of each other. Such a feature enables efficient implementation using SIMD units, GPUs or, in the case of FPGAs, embedded multipliers.
- Memory requirements. In many cases, execution speed may be limited by memory access latency. Ideally, an algorithm should require only a small amount of memory to execute. This enables one to keep all the data in the registers and L1 cache, which allow for the fastest access.
- Pipelining capability. Modern computing platforms owe their high potential speed to pipelining. Loosely speaking, pipelining means that one can start an operation (e.g. multiplication) once every clock cycle, but the results of operation are not available until several clock cycles later (latency). To illustrate potential caveats involving pipelining, consider a simple equation

$$y = ab + c$$

and assume that computational latency of both multiplication and adding is 5 cycles. This means that computing y will take as many as 10 cycles (and not 2 or 6 cycles), because it is necessary to compute result of multiplication prior to starting addition.

- Absence of feedback loops. Many adaptive algorithms involve feedback loops, e.g. to adjust adaptive weights. When feedback is present it is necessary to finish all computations from the previous phase before proceeding with the next. Feedback loops are undesirable because they usually make it hard or impossible to use pipeline efficiently. If feedback loops are present, it is desirable that adaptive coefficients are updated on a block (rather than sample) basis, i.e. once per block of data.
- Numerical stability. Some algorithms are known to possess poor numerical properties. In such a case it may be necessary to use floating-point arithmetic (which usually rules out FPGAs as a computing platform) and implement some additional safeguards.

In the sequel we will screen several well known adaptive filtering algorithms for these features.

3.0 ADAPTIVE FILTERING

3.1 Least squares methods

Denote by $x(t)$ the reference signal and let

$$y(t) = \sum_{k\tau=0}^{K-1} w_k x(t-k) + n(t) \tag{2}$$

denote the echo signal, consisting of delayed, scaled, and phase-shifted replicas of the reference, corrupted by a wideband noise $n(t)$. Eq. (2) can be written in the form

$$y(t) = \varphi^T(t)\boldsymbol{\theta} + n(t), \tag{3}$$

where $\varphi^T(t) = \begin{bmatrix} x(t) & x(t-1) & \dots & x(t-K+1) \end{bmatrix}$ and $\boldsymbol{\theta} = \begin{bmatrix} w_0 & w_1 & \dots & w_{K-1} \end{bmatrix}^T$. The least squares adaptive filter is defined as

$$y'(t) = y(t) - \varphi^T(t)\hat{\boldsymbol{\theta}},$$

where $\hat{\boldsymbol{\theta}}$ is the vector of adaptive weights, chosen so as to minimize the following cost criterion

$$J(\hat{\theta}) = \sum_{t=0}^{T-1} \left| y(t) - \varphi^T(t)\hat{\boldsymbol{\theta}} \right|^2.$$

The solution of the problem at hand is

$$\hat{\boldsymbol{\theta}} = \left(\boldsymbol{\Phi}^H\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^H\mathbf{y},$$

where $\mathbf{y} = \begin{bmatrix} y(0) & y(1) & \dots & y(T-1) \end{bmatrix}$ and

$$\boldsymbol{\Phi} = \begin{bmatrix} \varphi^T(0) \\ \varphi^T(1) \\ \vdots \\ \varphi^T(T-1) \end{bmatrix}.$$

Least squares is a purely block algorithm. This means that it can be parallelised and pipelined quite effectively. What can be criticized in the algorithm is that a large amount of operations is necessary to compute $\left(\boldsymbol{\Phi}^H\boldsymbol{\Phi}\right)$ and then to solve the system

$$\left(\boldsymbol{\Phi}^H\boldsymbol{\Phi}\right)\hat{\boldsymbol{\theta}} = \boldsymbol{\Phi}^H\mathbf{y}. \tag{4}$$

However, inspection of $\left(\mathbf{\Phi}^H\mathbf{\Phi}\right)$ reveals that (for large T) it is nearly Toeplitz – the formulas describing elements of $\left(\mathbf{\Phi}^H\mathbf{\Phi}\right)$ lying on the same diagonal are nearly identical. This can possibly be taken advantage of when computing $\left(\mathbf{\Phi}^H\mathbf{\Phi}\right)$.

Furthermore, one could try to reduce computational complexity even further by replacing $\left(\mathbf{\Phi}^H\mathbf{\Phi}\right)$ in (4) with a Toeplitz matrix $\mathbf{R}$ made up of, e.g. entries from the first row of $\left(\mathbf{\Phi}^H\mathbf{\Phi}\right)$. Not only that computing $\mathbf{R}$ is simpler than computing $\left(\mathbf{\Phi}^H\mathbf{\Phi}\right)$, this technique also enables one to use fast solvers for Toeplitz systems [9]. Note however, that if $\mathbf{R}$ is ill-conditioned, this may lead to large errors of the modified solution.

3.2 Least mean squares methods

A very popular implementation of adaptive filter is based on the least mean squares (LMS) approach

$$\hat{\boldsymbol{\theta}}(t) = \hat{\boldsymbol{\theta}}(t-1) + \mu\varphi(t)\left[y(t) - \varphi^H(t)\hat{\boldsymbol{\theta}}(t-1)\right], \tag{5}$$

where

$$\varphi(t) = \left[x(t) \quad x(t-1) \quad \ldots \quad x(t-K+1)\right]^T,$$

and $\mu > 0$ is a small gain. The output of the filter is

$$y'(t) = y(t) - \varphi^H(t)\hat{\boldsymbol{\theta}}(t)$$

or (a `free' variant)

$$y'(t) = y(t) - \varphi^H(t)\hat{\boldsymbol{\theta}}(t-1).$$

It is well known that convergence speed of LMS depends heavily on the eigenvalues of covariance matrix of the signal $x(t)$ [13]. However, when $y'(t)$ is of interest, slow convergence in weakly excited directions is less of an issue because these directions are actually not very pronounced in $y(t)$.

LMS is quite cheap in terms of number of operations and memory footprint. However, it involves feedback loop in the update equation (5) which can considerably complicate its implementation, especially on platforms such as FPGA. To mitigate this issue the so-called delayed LMS may be used

$$\hat{\boldsymbol{\theta}}(t) = \hat{\boldsymbol{\theta}}(t-1) + \mu\varphi(t-\tau)[y(t-\tau) - \varphi^H(t)\hat{\boldsymbol{\theta}}(t-\tau)]. \tag{6}$$

Such a form gives one considerably more time to compute the correction term. Furthermore, for small values of μ properties of delayed LMS are similar to (5). Poltmann [10] describes a procedure which allows one to convert the delayed LMS into the standard LMS at a price of slightly increasing computational cost.

Another important variant of LMS is the block version

$$\hat{\mathbf{\theta}}(Bt) = \hat{\mathbf{\theta}}(Bt - B) + \mu \mathbf{\Phi}(Bt)[\mathbf{y}(Bt) - \mathbf{\Phi}^H(Bt)\hat{\mathbf{\theta}}(Bt - B)], \quad (7)$$

where B is the block size (usually B is equal to the size of $\hat{\mathbf{\theta}}$ and a power of two), $\mathbf{y}(Bt) = \begin{bmatrix} y(Bt) & y(Bt-1) & \dots & y(Bt-B+1)^T \end{bmatrix}$ and

$$\mathbf{\Phi}(Bt) = \begin{bmatrix} \varphi(Bt) & \varphi(Bt-1) & \dots & \varphi(Bt-B+1) \end{bmatrix}.$$

The block version is particularly attractive, because the multiplications $\mathbf{\Phi}^H(Bt)\hat{\mathbf{\theta}}(Bt-B)$ and $\mathbf{\Phi}(Bt)\mathbf{e}(Bt)$,where $\mathbf{e}(Bt) = \mathbf{y}(Bt) - \mathbf{\Phi}^H(Bt)\hat{\mathbf{\theta}}(Bt-B)$, are simply convolutions. This means that the most time-consuming parts of (7) may be implemented very efficiently using FFT. When the overlap-save method is used to implement convolutions, (7) requires 5 FFTs of size $2B$ per block of data. For large filter sizes the FFT-based implementation can be more than ten times faster than (5)-(7). A delayed variant of block LMS, suitable for pipelined implementation is described in [11].

3.3 Lattice adaptive filters

Block diagram of a lattice filter is shown in Figure 2. The filter consists of two distinct parts. The top part of the filter decomposes the signal $x(t)$ into forward and backward prediction errors, using so-called reflection coefficients $\Gamma_k, k = 1,2,...,K$

$$\begin{aligned} f_k(t) &= f_{k-1}(t) + \Gamma_k^* b_{k-1}(t-1) \\ b_k(t) &= b_{k-1}(t-1) + \Gamma_k f_{k-1}(t). \end{aligned} \quad (8)$$

The lattice is initialized as

$$f_0(t) = b_0(t) = x(t).$$

The bottom part of the filter performs the cleaning – weighted backward prediction errors are subtracted from the input signal so as to minimize the power of the output signal.

$$e_{k+1}(t) = e_k(t) - w_k(t) b_k(t), \quad (9)$$

where $e_0(t) = y(t)$.

Lattice filters are often praised for their ability to grow and shrink as needed. The size of the lattice may be freely adjusted on-line, i.e. when the filter is running. However, note that in case of an embedded implementation, the order of the filter is likely to be constant or bounded. This means that this particular feature may be of little importance.

The optimal (in the mean-sqared sense) reflection coefficients and weight are defined using ensemble means

$$\begin{aligned}\Gamma_k &= \frac{E[b_{k-1}(t-1)\, f_{k-1}^*(t)]}{E[|\, f_{k-1}(t)\,|^2]} \\ w_k &= \frac{E[y_k(t-1) b_k^*(t)]}{E[|\, b_k(t)\,|^2]}.\end{aligned} \tag{10}$$

In practice the ensemble means are unknown and the coefficients of the filter must be adapted in a different way. One of the most straightforward solutions is based on replacing ensemble means with statistical ones [5]

$$\begin{aligned}\Gamma_k &= \frac{\sum b_{k-1}(t-1)\, f_{k-1}^*(t)}{\sum |\, f_{k-1}(t)\,|^2} \\ w_k &= \frac{\sum y_k(t-1) b_k^*(t)}{\sum b_k(t)\,|^2}.\end{aligned} \tag{11}$$

Such an approach has several strong points. Computation of forward and backward prediction error signals, as well as the next reflection coefficient Γ_k, and the weight b_k may be performed in parallel. Furthermore, any of these operations may be distributed among several cores.

Unfortunately, the statistical mean approach is not free of drawbacks. Firstly, it requires a rather large amount of memory to keep the whole signal recording. Secondly, the stages of the filter must be executed in sequence. Note that, prior to executing equations for the k-th stage, one must first compute the corresponding reflection coefficient, which is done in the $(k-1)$-th stage. This limits the amount of parallelism in the algorithm. Furthermore, it is necessary to keep the computed signals $f_k(t), b_k(t)$ in memory for cleaning. Since the processed signal are usually long, this may lead to poor usage of cache memory.

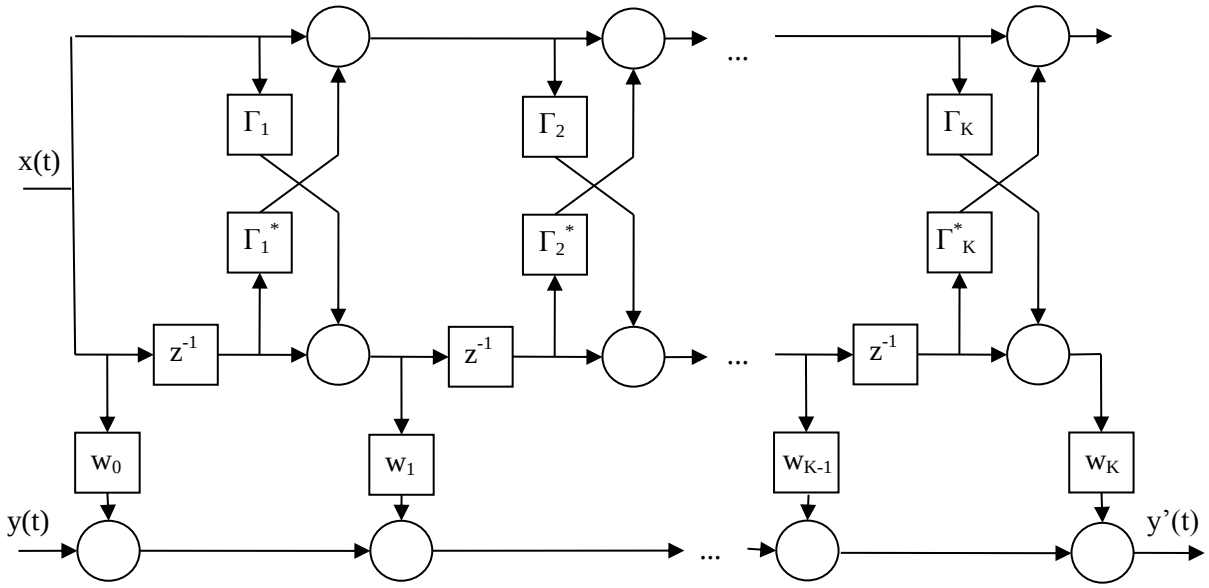

Figure 2: Lattice filter.

Other adaptation algorithms include, e.g. the least squares lattice (LSL) algorithm or the gradient adaptive lattice (GAL) algorithm [13]. The former is a rather heavyweight algorithm, which includes many divisions, sequential processing and feedback loop. It is also well known that certain implementations of LSL suffer from numerical instability. This makes LSL rather poorly suited for radar signal processing.

GAL is considerably lighter than LSL. The algorithm reads

$$
\begin{aligned}
f_k(t) &= f_{k-1}(t) + \hat{\Gamma}_k^*(t) b_{k-1}(t-1) \\
b_k(t) &= b_{k-1}(t-1) + \hat{\Gamma}_k(t) f_{k-1}(t) \\
\xi_{k-1}(t) &= \beta \xi_{k-1}(t-1) + (1-\beta)(|f_{k-1}(t)|^2 + |b_{k-1}(t-1)|^2) \\
\hat{\Gamma}_k(t+1) &= \hat{\Gamma}_k(t) - \frac{1}{\xi_{k-1}(t)}[f_{k-1}^*(t) b_k(t) + b_{k-1}(t-1) f_k^*(t)].
\end{aligned}
\tag{12}
$$

The above recursion implements only the lattice part. To complete the joint process estimation, it should be extended with a LMS-like part for adapting the weights w_k.

Note that GAL includes feedback loop, divisions, and a considerable number of sequential operations. All these features make the algorithm difficult to implement efficiently.

3.2 Tracking behaviour of adaptive filters

We will discuss tracking behaviour of adaptive filters using LMS as an example. Let

$$y(t) = \varphi^H(t)\boldsymbol{\theta}(t) + n(t), \tag{13}$$

where $\boldsymbol{\theta}(t)$ is the true, slowly time-varying, vector of parameters. For small values of adaptation gain μ, the estimated weight vector $\hat{\boldsymbol{\theta}}(t)$ `wanders' around its statistical expectation, $\mathrm{E}[\hat{\boldsymbol{\theta}}(t)]$.

Recall (5) and apply expectation to both sides. Neglecting correlation between $\varphi^H(t)$ and $\hat{\boldsymbol{\theta}}(t-1)$ one obtains

$$\mathrm{E}[\hat{\boldsymbol{\theta}}(t)] = \mathrm{E}[\hat{\boldsymbol{\theta}}(t-1)] - \mu\mathbf{R}_\varphi \mathrm{E}[\hat{\boldsymbol{\theta}}(t-1)] + \mu\mathbf{R}_\varphi \boldsymbol{\theta}(t). \tag{14}$$

where $\mathbf{R}_\varphi = \mathrm{E}[\varphi(t)\varphi^H(t)]$. Denote by λ_i the i-th eigenvalue of $\mathbf{R}_\varphi$ and by $\mathbf{v}_i$ the corresponding eigenvector. Let $\boldsymbol{\theta}_i(t) = \mathbf{v}_i^H\boldsymbol{\theta}(t)$ and $\hat{\boldsymbol{\theta}}_i(t) = \mathbf{v}_i^H\mathrm{E}[\hat{\boldsymbol{\theta}}(t)]$. Then

$$\hat{\boldsymbol{\theta}}_i(t) = (1-\mu\lambda)\hat{\boldsymbol{\theta}}_i(t-1) + \mu\lambda_i\boldsymbol{\theta}_i(t), \tag{15}$$

i.e. the behaviour of LMS can be explained as first order filtering of $\boldsymbol{\theta}_i(t)$. It follows that one can introduce a concept of frequency response of LMS, which takes the form

$$G_i(e^{-j\omega}) = \frac{\mu\lambda_i}{1-(1-\mu\lambda)e^{-j\omega}}. \tag{16}$$

Eq. (16) may be used to assess tracking error and phase lag of the LMS adaptive filter.

However, in case of radar signal processing, the tracking error frequency response of $\boldsymbol{\theta}_i(t) - \hat{\boldsymbol{\theta}}_i(t)$ is of more interest. It takes the form

$$H_i(e^{-j\omega}) = 1 - G_i(e^{-j\omega}) = \frac{(1-\mu\lambda)(1-e^{-j\omega})}{1-(1-\mu\lambda)e^{-j\omega}} \tag{17}$$

and is shown in Figure 3 for $\mu\lambda_i = 0.05$. As expected, the LMS provides a notch around zero Doppler. However, note that the range of frequencies with very high attenuation is very narrow. Increasing μ could widen the notch, but this could adversely affects higher frequencies, where targets are present.

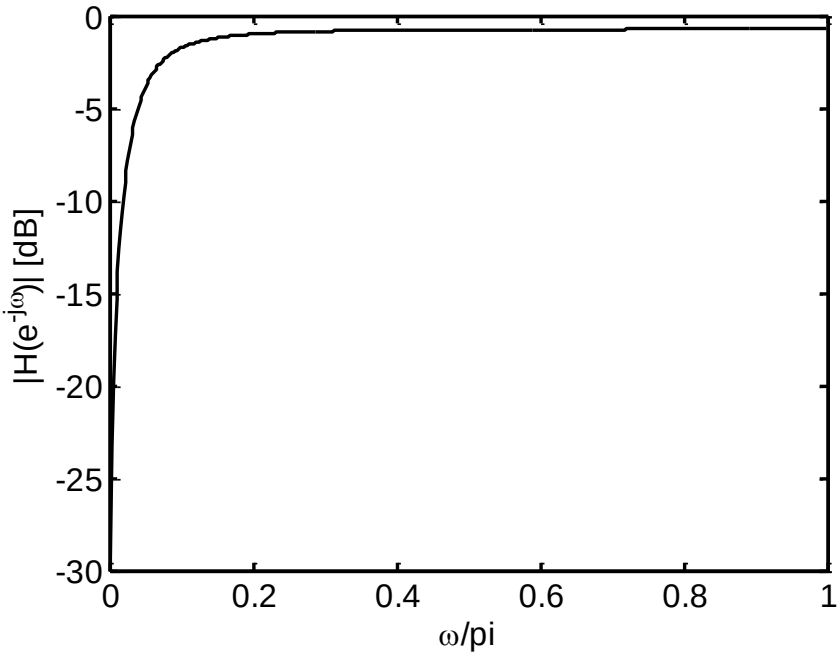

Figure 3: Tracking error frequency response of LMS.

Eq. (11) suggest another way to widen the notch of LMS. This can be achieved by postfiltering $\hat{\boldsymbol{\theta}}_i(t)$

$$\bar{\boldsymbol{\theta}}_i(t) = F_i(e^{-j\omega})\hat{\boldsymbol{\theta}}_i(t), \tag{18}$$

where $F_i(e^{-j\omega})$ is chosen so as to improve the resulting error frequency response. The filtering technique may also be extended to cancelation of moving targets, see [12].
The concept of frequency response may be generalized to block algorithms, such as the LS approach. The estimator (4) yields a value which is close to the averaged true parameter vector

$$\mathrm{E}[\hat{\boldsymbol{\theta}}] = \frac{1}{T}\sum_{\tau=0}^{T-1}\boldsymbol{\theta}(t-\tau). \tag{19}$$

This means that its frequency response reads

$$G(e^{-j\omega}) = \frac{e^{-j\omega(T-1)/2}}{T}\frac{\sin(\omega T/2)}{\sin(\omega/2)}. \tag{20}$$

The resulting error frequency response is shown in Figure 4 (It was assumed that $T = 20$). The ripples present in the response are actually less important than it might seem, because crossambiguity of the cleaned signal is typically computed at the points where $H_i(e^{-j\omega}) = 1$.

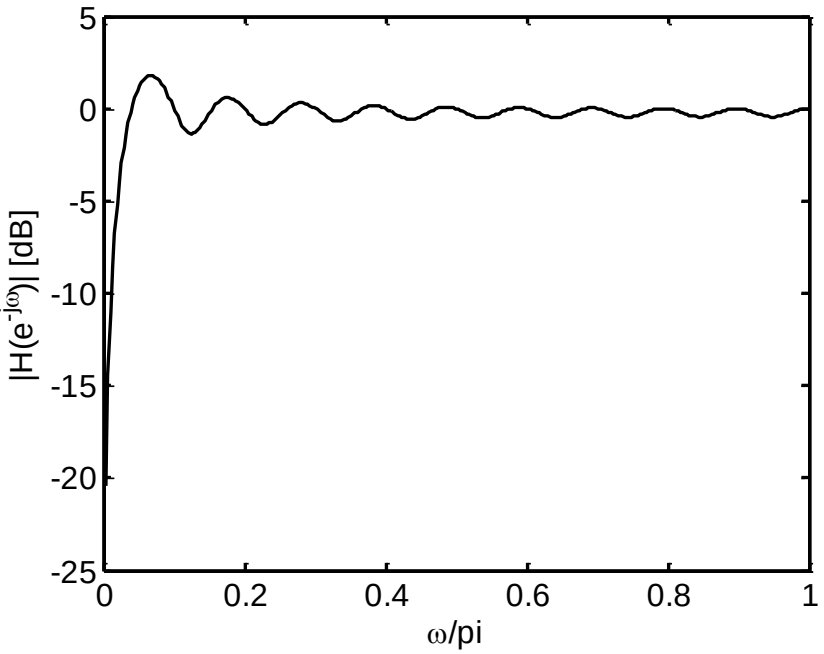

Figure 4: Typical tracking error frequency response of a block algorithm.

4.0 CROSSAMBIGUITY

4.1 Time domain implementation

The most straightforward design is direct implementation of the crossambiguity formula

$$\chi(\tau,\omega) = \sum_{t=0}^{T-1} y'(t)x^*(t-\tau)e^{-j\omega t}\,, \tag{21}$$

which requires $2T$ complex multiplications for each point on the grid (τ,ω).

Several easy improvements can be made to this approach. First, note that for fixed τ and $\omega = 2k\pi/T$, $\chi(\tau,\cdot)$ may be interpreted as the DFT of the sequence $y'(t)x^*(t-\tau)$. Furthermore, since useful information resides only in the low frequency part of the spectrum of $y'(t)x^*(t-\tau)$, a lowpass filter and a decimator may be applied prior to spectral analysis. Typically, this will reduce a stream of data applied on the FFT processors by 2 or more orders of magnitude, thus making the cost of the FFTs negligible.

Out of possible solutions, the CIC decimator is particularly interesting. When CIC decimator is used, the crossambiguity formula is implemented as

$$\chi(\tau,\omega)=\sum_{b=0}^{B-1}\left[\sum_{t=0}^{L-1}y'(bL+t)x^{*}(bL+t-\tau)\right]e^{-j\omega bL}, \tag{22}$$

where L denotes CIC order. Note that the process of computing $\chi(\tau,\omega)$ was decomposed into correlation (inner sum) and DFT (the outer sum). An important feature of the above formula is that the correlation part may now be implemented via FFT. This will be explained in more details in the next subsection.

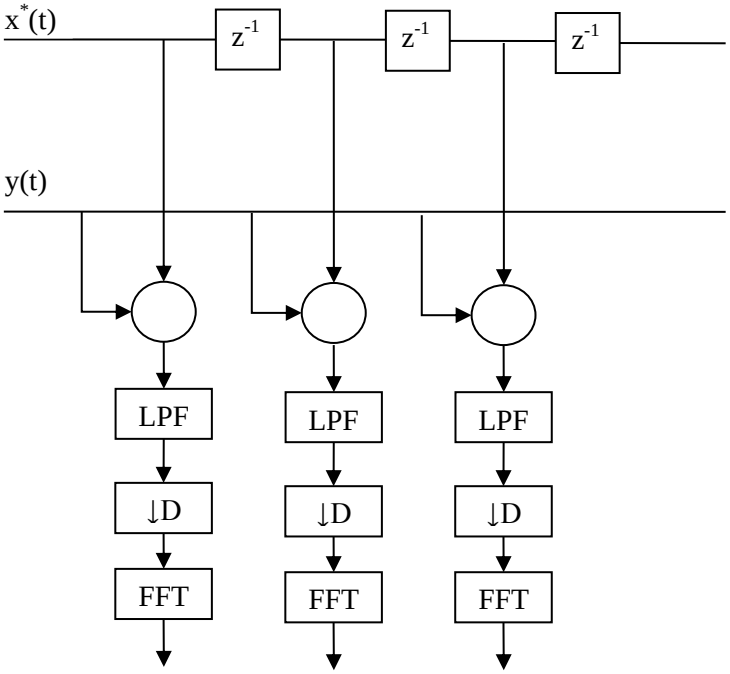

Figure 5: Time domain implementation of the crossambiguity computer employing sampling rate reduction.

4.2 Fast algorithm via FFT

Consider the product

$$z(\tau) = \sum_{t=0}^{L-1} y'(t) x^*(t-\tau). \tag{23}$$

Note that

$$z(\tau) = \sum_{t=0}^{L-1} y'(t) x^*(-(\tau - t)), \tag{24}$$

i.e. $z(\tau)$ is a convolution of $y'(t)$ and $x^*(-t)$. This means that very efficient FFT based algorithms for computing convolution may be applied to compute $z(\tau)$ for several consecutive values of τ. Since it holds that

$$FFT[x^*(-t)] = FFT^*[x(t)],$$

the following algorithm, based on the overlap-save method, may be obtained.

1. Divide the signals into B=T/L blocks of length L.

$$\mathbf{x}(b) = \begin{bmatrix} x(bL) & x(bL-1) & \dots & x(bL-L+1) \end{bmatrix}^T$$
$$\mathbf{y}(b) = \begin{bmatrix} y(bL) & y(bL-1) & \dots & y(bL-L+1) \end{bmatrix}^T.$$

   For efficiency, L should be chosen to be a power of two.

2. For each block, form the superblocks of size 2L

$$\mathbf{X}(b) = \begin{bmatrix} \mathbf{x}^T(b-1) & \mathbf{x}^T(b) \end{bmatrix}^T$$
$$\mathbf{Y}(b) = \begin{bmatrix} \mathbf{0}^T & \mathbf{y}^T(b) \end{bmatrix}^T.$$

3. Compute FFTs of both superblocks.

$$\mathbf{X}_F(b) = FFT\left[\mathbf{X}(b)\right]^T$$
$$\mathbf{Y}_F(b) = FFT\left[\mathbf{Y}(b)\right]^T.$$

4. Multiply FFTs pointwise, taking into account that $\mathbf{X}_F(b)$ must be conjugated.

$$\mathbf{Z}_F(b) = \mathbf{Y}_F(b) \otimes \mathbf{X}_F^*(b).$$

5. Compute inverse FFT of $\mathbf{Z}_F(b)$ and discard its meaningless half $\boldsymbol{\zeta}(b)$.

$$\mathbf{Z}(b) = \left[\mathbf{z}^T(b) \quad \boldsymbol{\zeta}^T(b)\right]^T,$$

where $\mathbf{z}(b) = [z(0)\, z(1) \ldots z(L-1)]^T$.

The block diagram of the algorithm is shown in Figure 6. Observe that it is a fully pipelined design. Since FFT cores are readily available from FPGA manufacturers, implementation of the algorithms is rather straightforward. For instance, the core provided by Xilinx [14] allows one to compute FFTs of the signal sampled at an arbitrary clock rate in real-time. The major limitation of the algorithm is the FFT length, which must be equal twice the block length. This limits signal sampling rate to half of the FPGA clock rate, unless certain extension to the implementation, such as doubling the number of FFTs/IFFTs are made.

Remark: The correlator block should be followed by a Doppler processor to complete the process of computing crossambiguity.

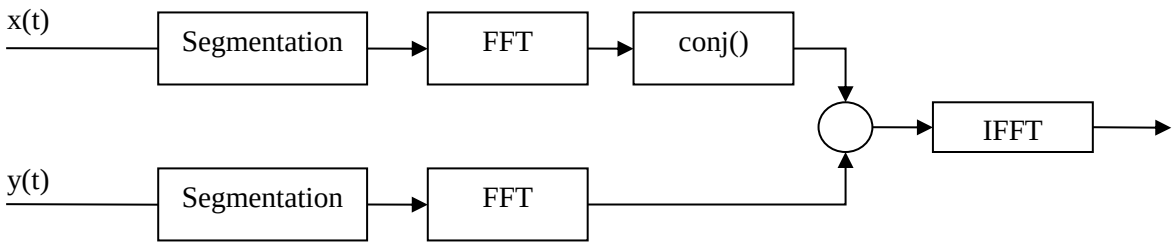

Figure 6: Fast implementation of correlation based on FFT.

4.3 Frequency response of the fast algorithm

Let $y'(t) = e^{j\omega_0 t} x(t)$. Then

$$\left|\mathrm{E}[z(0)]\right|^2 = \left|\sum_{t=0}^{L-1} e^{j\omega_0 t}\sigma_x^2\right|^2 = \left[\frac{\sin(\omega_0 L/2)}{\sin(\omega_0/2)}\sigma_x^2\right]^2. \tag{25}$$

which means that a loss can be expected for high values of Doppler. Figure 7 shows dependence of loss on target velocity for a radar operating at a 10 GHz carrier with 10 MHz sampling frequency and block size L=1024. Under the assumption that losses should not exceed 1 dB, the design is suitable for targets with speeds not exceeding 38 m/s. Such a value may not be satisfactory for certain applications, and one could be forced to use shorter blocks.

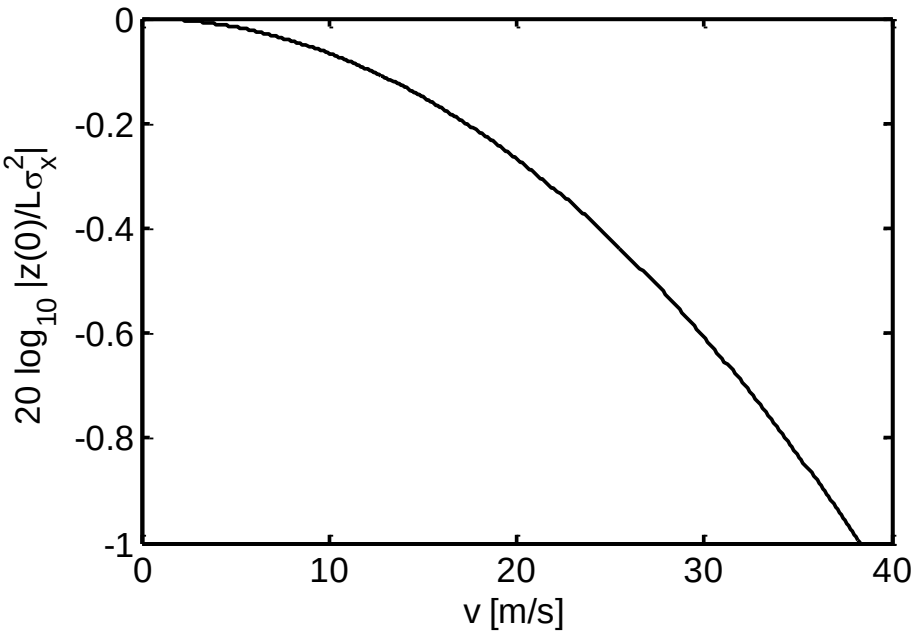

Figure 7: Dependence of loss on target velocity for X-band radar and L=1024.

In such a case, the implementation shown in Figure 8 may be useful. This variant trades block length for additional multipliers, memory and IFFT engines, but avoids adding extra FFTs to the design.

5.0 CONCLUSIONS

Several options for adaptive filtering and computation of crossambiguity function were evaluated in terms of their suitability for passive radars with high sampling rate. We compared their computational complexity, memory usage and suitability for pipelined implementation.

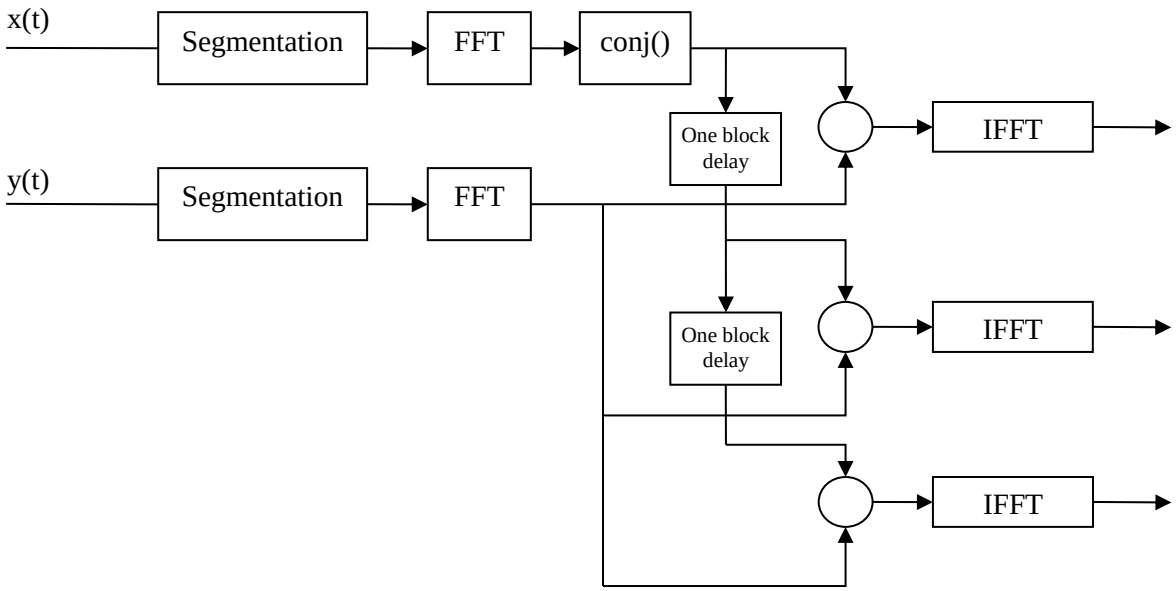

Figure 8: Fast implementation of correlation based on FFT for short block sizes.

REFERENCES

[1] P.E. Howland, D. Maksimiuk, G. Reitsma, FM radio based bistatic radar, IEE Proceedings - Radar, Sonar and Navigation, vol. 152, no. 3, pp. 107-115, 2005.

[2] F. Berizzi, D. Petri, A. Capria, M. Martorella, High range resolution DVB-T Passive Radar, Proc. 2010 European Radar Conference (EuRAD 2010), 109-112, 2010.

[3] R. Zemmari, U. Nickel, W.D. Wirth, GSM passive radar for medium range surveillance, Proc. 2009 European Radar Conference, 2009 (EuRAD 2009). 49-52, 2009.

[4] Guangjie Gao, Qing Wang, Chunping Hou, Power budget and performance prediction for WiMAX based passive radar, Proc 6th International Conference on Pervasive Computing and Applications (ICPCA 2011), 517-520, 2011.

[5] M. Malanowski, K. Kulpa, Detection of Moving Targets With Continuous-Wave Noise Radar: Theory and Measurements, IEEE Transactions on on Geoscience and Remote Sensing, vol. 50, no. 9, pp. 3502-3509, 2012.

[6] D.W. O'Hagan, and C.J. Baker, H.D. Griffiths, M. Inggs, R. Lord, N. Morrison, Passive Radar Tracking, The Institution of Engineering and Technology Seminar onThe Future of Civil Radar, pp. 57-67, 2006.

[7] G. Battistelli, L. Chisci, S. Morrocchi, F. Papi, A. Farina, A. Graziano, Robust Multisensor Multitarget Tracker with Application to Passive Multistatic Radar Tracking, vol. 48, no. 4, pp. 3450-3472, 2012.

[8] M. Malanowski, K. Kulpa, R. Suchozebrski, Two-stage tracking algorithm for passive radar , 12th International Conference on Information Fusion (FUSION '09), pp. 1800-1806, 2009.

[9] R. P. Brent, Old and new algorithms for Toeplitz systems, Proceedings SPIE, vol. 975, Advanced Algorithms and Architectures for Signal Processing III (edited by Franklin T. Luk), pp. 2-9, 1989.

[10] R.D. Poltmann, Conversion of the delayed LMS algorithm into the LMS algorithm, IEEE Signal Processing Letters, vol. 2, no. 12, 1995.

[11] B. K. Mohanty, Delayed block LMS algorithm and concurrent architecture for high-speed implementation of adaptive FIR filters, 2008 IEEE Region 10 Conference (TENCON 2008), 2008.

[12] M. Meller, Cheap Cancellation of Strong Echoes for Digital Passive and Noise Radars, IEEE Transactions on Signal Processing, vol. 60, no. 5, pp. 2654-2659, 2012.

[13] S. Haykin, Adaptive Filter Theory, Prentice Hall, 1991.

[14] Xilinx DS260 LogiCORE IP Fast Fourier Transform v7.1, Xilinx.