MASTER

Algorithms and implementation of an adaptive filter for a quality surveillance system

Gebeyehu, M.

Award date:
1997

TECHNISCHE UNIVERSITEIT EINDHOVEN

FACULTEIT ELEKTROTECHNIEK

LEERSTOEL Signaalverwerking

SIGNAL PROCESSING

Algorithms and implementation of an adaptive filter for a quality surveillance system

door

M. Gebeyehu

ESP-18-97

Verslag van een afstudeeronderzoek
verricht binnen de leerstoel ESP
onder leiding van dr.ir. A.C. den Brinker
in de periode januari 1997 - oktober 1997

Eindhoven, 16 oktober 1997

Abstract

In this thesis the replacement of the analog front end of a quality surveillance system by a digital one is studied. The requirements for this digital system are that it has a performance at least as good as the existing system and that it is more flexible.

In order to attain these goals a signal model is proposed and confronted with the actual data. The existing system is translated to the digital domain. It is shown that adaptive systems can be used but only with due care since the signal model does not agree with the signal model for which adaptive systems are intended. A theoretical analysis of one-parameter adaptive filters is presented extending the existing analysis to the case of nonwhite input and/or reference signals. Finally, a DSP implementation is established as a discrete counterpart for the existing system. Using a DSP-system for the implementation, the proposed system is inherently more flexible since the software can be easily adapted.

Contents

Chapter 1

Introduction

This introduction provides an overview of the contents of this thesis and of the principles involved in the implementation of an adaptive filter. The quality surveillance system for yarn produced by BARCO Automation (Kortrijk, Belgium) has been designed for detecting the color of a thread. The measurement set-up is shown in Fig. 1.1.

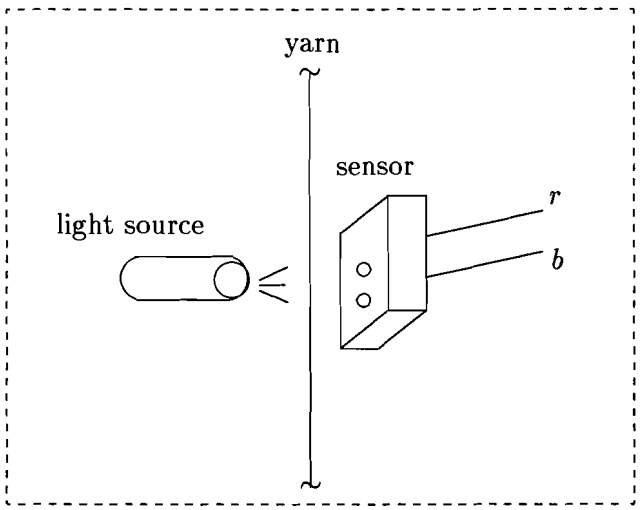

Figure 1.1: Measurement set-up of the quality surveillance system.

The light source with a certain color spectrum is used for lighting the thread and the sensors are used to measure the light reflected by the thread. The output of these sensors are electrical signal (b and r) which are proportional to the reflected light intensities in the blue range and red range, respectively. The ratio of these color components is constant in the absence of variation in the intensity of the light source or in the thickness of the thread. So our object of study is to analyze these color components in the presence of above mentioned variation and to estimate the ratio by means of an adaptive algorithm. The project also includes implementing a digital based quality surveillance system. This digital based system is expected to have more flexibility and an improved performance compared with the existing analog quality surveillance system.

The digital quality surveillance system is implemented on the TMS320C40 ('C40) chip of

Texas Instruments. The TMS320C40 has been designed to provide efficient implementation of many common digital signal processing (DSP) algorithms and its instruction is exceptionally well-suited to DSP applications. Important characteristics of the 'C40 that contribute to its high performance are, pipelining (i.e, the overlapping operations of the fetch, decode, read and execute labels of a basic instruction), concurrent I/O and CPU operation. So exploiting these and more characteristics of 'C40 is a good choice which many firms are unable to exploit due to lack of expertise/time.

We begin Chapter 2 by introducing the mathematical formulation of the problem. In particular we introduce here our assumption of the signals in the noise environment. The sources of noise will also be discussed here. In Chapter 3 we will discuss the existing quality surveillance system and in Chapter 4 we analyze the blue signal (b) and the red signal (r) by means of sampled experimental data, taken from the sensor. We will also consider the estimated functional relation and the distribution of this experimental data.

In Chapter 5 we introduce adaptive filters based on a gradient search technique for minimizing a quadratic performance function (LMS) and on minimizing an exponential weighted sum of squared error (RLS). The simulation results of these algorithms on the experimental data discussed in Chapter 4 are presented in Chapter 6. The simulation programs were developed in MatLab.

Chapter 7 discusses the implemented C program for the digital quality surveillance system with 'C40. A general description of the 'C40 will also be given here. Chapter 8 contains an evaluation of the designed digital quality surveillance system and Chapter 9 draws conclusions and recommendations.

Chapter 2

Problem description

2.1 Signal definitions and assumptions

The intensities of the light reflected by a piece of thread of the yarn in the red and blue spectrum are considered. Ideally, the reflected light intensities are constant. However, in the presence of variations in the intensity of the light source or in the thickness of the thread, the reflected light intensities in the blue and red range will vary. The ratio of these reflected light intensities will nevertheless be constant. The sources of the variations are considered as stochastic quantities. The intensities of the reflected light in the red and blue range are called r'_0 and b'_0, respectively, and are stochastic quantities as well. The working point is defined as $(\overline{r'_0}, \overline{b'_0})$ where a bar denotes the (time-)average of a stochastic quantity. We introduce the color coefficient α_0 as

$$\alpha_0 = b'_0/r'_0. \tag{2.1}$$

Under the given assumptions, plotting b'_0 versus r'_0 would result in occurrences in the $r-b$ plane exactly on the line defined by α_0 and near the working point.

There are several sources of noise which we take into account. First of all there is measurement noise, e.g., nonperfect alignment of thread and sensor, noise in the sensor itself and quantization noise in digital representation of the signal. Also, small color deviations are allowed (depending on the specified quality of constancy of the color of the thread). These deviations on the measured light intensities are treated as stochastic quantities and are called n_1 and n_2 for the red and blue range, respectively. Furthermore, we assume that $E\{n_i\} = 0$ and

$$\begin{aligned} E\{n_1^2\} &\ll E\{[r'_0 - \overline{r'_0}]^2\}, \\ E\{n_2^2\} &\ll E\{[b'_0 - \overline{b'_0}]^2\}. \end{aligned}$$

(Note that if power of the noise n_1 and n_2 would be larger than those of r'_0 and b'_0 we could have restricted ourselves to the working point only.) Plotting $r'_0 + n_1$ versus $b'_0 + n_2$ in the $r-b$ plane would result in an ellipsoidal distribution of the points around the working point with the main axis of the ellipse on the line defined by α_0.

As a last source of deviations of the ideal case we have the occurrence of a different color of yarn, and of dark grains in the cotton. These deviations are also considered as stochastic quantities and are denoted s_1 and s_2 for the red and blue color range, respectively. The dark

grains are inevitable, but different colored pieces of yarn have to be detected (and disposed of). It is assumed that these deviations are zero-mean, large i.e.

$$E\{s_1^2\} \gg E\{[r_0' - \overline{r_0'}]^2\},$$
$$E\{s_2^2\} \gg E\{[b_0' - \overline{b_0'}]^2\}.$$

and, furthermore, that they are relatively short lasting.

The actual signals r and b that we dispose of are thus assumed to be

$$r = r_0 + n_1 + s_1,$$
$$b = b_0 + n_2 + s_2,$$

since the measurement system provides us with high-pass filtered versions of the reflected light intensities (Chapter 3) and

$$r_0 = r_0' - \overline{r_0'},$$
$$b_0 = b_0' - \overline{b_0'}.$$

The reason for highpass filtering is that the sensors do not only measure the light reflected by the thread but also reflections from the (steady) background.

For convenience of subsequent analysis, the two kind of deviations of the ideal behavior are often taking as a single noise source, i.e., we define

$$n_r = n_1 + s_1,$$
$$n_b = n_2 + s_2.$$

Plotting the signal r versus b would thus presumably lead to an ellipsoidal distribution in the $r-b$ plane around the point $(0,0)$ with the main axis defined by α_0 and with (occasionally) a number of outliers.

It is assumed that all covariances of the fundamental variables r_0, b_0, n_1, n_2, s_1 and s_2 are equal to zero with, naturally, the exception of $E\{r_0 b_0\}$ and, possibly, $E\{s_1 s_2\}$. The probability density functions (pdf) of r_0 and b_0 are identical (except for scaling). Similarly, it is expected that the pdf of $r_0 + n_1$ and $b_0 + n_2$ are nearly identical to those of r_0 and b_0 since by definition n_1 and n_2 are small. If the pdfs of r and b are not nearly identical, than it may be inferred that there is a component s_1 and/or s_2 interfering on the data (reversal of this statement is not possible).

2.2 Frequency domain description

In essence, the model of the signal generation that is proposed is given in Fig. 1.1, left side. We have the signals r_0 and b_0 functionally related by $r_0 = \alpha_0 b_0$ and the disturbances n_i and s_i. Available to measurement are the signals r and b (or r and b are the measured signals and the noise sources contain measurement noise as well). A similar model is the one given in the righthand side of Fig. 1.1 with $\beta_0 = 1/\alpha_0$.

In the frequency domain we can describe the stochastic signals by their power spectral density functions

$$S_{rr}(\Omega) = \mathcal{F}E\{r[k]r[k+n]\}$$
$$S_{bb}(\Omega) = \mathcal{F}E\{b[k]b[k+n]\}$$
$$S_{rb}(\Omega) = \mathcal{F}E\{r[k]b[k+n]\},$$

Figure 2.1: Signal models. The two signal models are identical if $\beta_0 = 1/\alpha_0$.

where $\mathcal{F}$ denotes the Fourier transformation. If there are no large disturbances ($s_1 = s_2 = 0$) than we expect that (except for scaling) $S_{rr} \propto S_{bb}$ (reversal of the statement is not possible). Furthermore,

$$\begin{aligned} S_{rb}(\Omega)/S_{bb}(\Omega) &\approx \text{constant} = \alpha_0 \\ S_{rb}(\Omega)/S_{rr}(\Omega) &\approx \text{constant} = \beta_0 \end{aligned}$$

If there are large disturbances than these ratios can still be frequency independent but it would imply that not only the disturbances are correlated but also functionally related in the same way as r_0 and b_0 (and therefore indistinguishable from the true color signals r_0 and b_0).

In general, if two signals are functionally related by some linear operator $x \rightarrow y$, than we have

$$\begin{aligned} S_{xy}(\Omega)/S_{xx}(\Omega) &= T_{xy}(e^{j\Omega}) \\ S_{yx}(\Omega)/S_{yy}(\Omega) &= T_{yx}(e^{j\Omega}) = T_{xy}^{-1}(e^{j\Omega}). \end{aligned}$$

and thus

$$C(\Omega) = \frac{S_{xy}(\Omega)S_{yx}(\Omega)}{S_{xx}(\Omega)S_{yy}(\Omega)} = 1. \tag{2.2}$$

The function C is referred to as the coherence function. However, if there is noise interfering than from C we can see which part of the power spectra of x and y can be described by a linear operator (i.e., for those frequencies where $C = 1$) and on which frequency parts of x and/or y the noise is interfering.

Chapter 3

Existing system

In this chapter we introduce the functional behavior of the existing analog adaptive mechanism. This introduction serves as the basic building blocks from which we can construct other adaptive algorithm with equivalent if not better performance. We also examine the relation of this existing system parameters to the most widely used adaptation method (LMS).

3.1 Scheme of the functional behavior of the analog system

Fig. 3.1 shows a block scheme of the adaptation algorithm used in the analog quality surveillance system with exception of the scaling factors of the input signals.

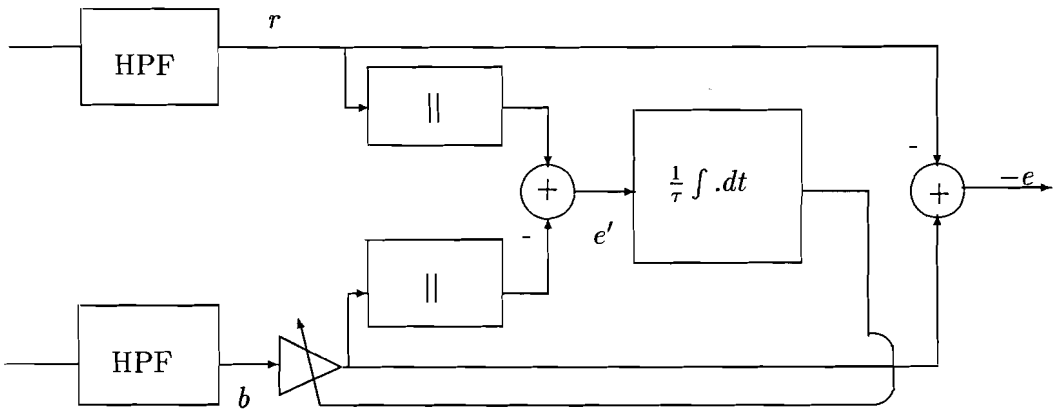

Figure 3.1: Block schema of the existing adaptive algorithm.

The input signals pass through a highpass filter (HPF), which is a first order filter with cut-off frequency at 0.85Hz, in order to filter out the DC-components (especially light reflected by the background), and enters the adaptive mechanism. The output of this circuit is the error signal e, which is given by $e = r - \alpha b$. This adaptive mechanism tries to estimate the color coefficient which is the ratio of the blue-contents and the red-contents. Due to the prefiltering it is not possible to determine this coefficient by simply dividing these two signals.

Fig. 3.2 shows a simplified block schema of the adaptation mechanism shown in Fig. 3.1. The output is the error signal e', which is given by $e' = |r| - \alpha|b|$. The input-output relation

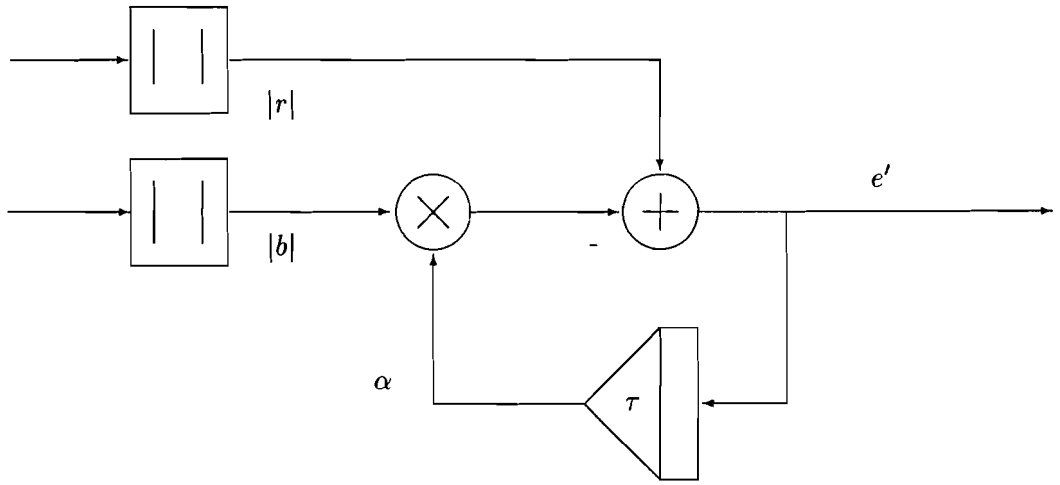

Figure 3.2: Simplified block schema of the existing adaptive mechanism.

of this system is given by the differential equation

$$\tau\dot{\alpha} + \alpha(t)|b(t)| = |r(t)| \tag{3.1}$$

with $\tau = 44ms$.
The signal e is the input to a decision making mechanism (digital) which judges if color deviations occur and if these pieces of thread have to be removed.

3.2 Discretized versions

In order to simulate the system, the amplitude is quantized into discrete steps and at the same time the signal is sampled at discrete time intervals. To avoid aliasing, the input signals spectrum is limited and its bandwidth is smaller than half the sampling frequency.

Discretization by numerical integration:
By applying the 'trapezium-rule' to (3.1), we obtain a difference equation:

$$\alpha[k] = \frac{|r[k]| + |r[k-1]|}{2\mu + |b[k]|} + \frac{2\mu - |b[k-1]|}{2\mu + |b[k]|}\alpha[k-1]$$

so, that

$$\alpha[k] = \alpha[k-1] + \frac{D}{\tau}\left(\frac{e'[k] + e'[k-1]}{2}\right). \tag{3.2}$$

Discretization by using forward differences:
By replacing the derivative by a forward difference in (3.1) we obtain:

$$\tau\frac{\alpha[k] + \alpha[k-1]}{D} = |r[k-1]| - \alpha[k-1]|b[k-1]| = e'[k-1]$$

so that

$$\alpha[k] = \alpha[k-1] + \frac{D}{\tau} e'[k-1] \tag{3.3}$$

where D is the sampling time.
In the first and third quadrant of the $b-r$ plane this equation yields:

$$\alpha[k+1] = \alpha[k] + \frac{D}{\tau} sgn(b[k])e[k] \tag{3.4}$$

In the second and fourth quadrant of the $b-r$ plane we have

$$\alpha[k+1] = \alpha[k] - \frac{D}{\tau} sgn(b[k])e[k]$$

Since the signal is primarily in the first and third quadrant, the update rule is almost always the one given in (3.4). The usual LMS algorithm has the form (see Section 5):

$$\alpha[k+1] = \alpha[k] + \mu b[k]e[k] = \alpha[k] + \mu|b[k]|sgn(b[k])e[k]$$

Comparing this with (3.4) we observe that adaption formula is similar if appropriate choice of the LMS parameter is made.

In order to get similar performances to the existing system with the LMS algorithm we take:

$$\mu = \frac{D}{\tau|b[k]|}$$

Assuming that the signal b follows a Gaussian distribution, we have

$$\overline{|b|} = \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}\sigma} |b| e^{\frac{-b^2}{2\sigma_b{}^2}} db = \sqrt{\frac{2}{\pi}}\sigma_b.$$

So, it is expected that the LMS algorithm will give results similar to the existing system.

$$\mu = \frac{D}{\tau} \frac{1}{0.8\sigma_b}. \tag{3.5}$$

3.3 Conclusions

- The existing system is almost LMS algorithm in first and third quadrate of the $b-r$ plane.
- The existing adaptive algorithm does not adapt in the right direction if the signal (b, r) is in the second or fourth quadrant of the $b-r$ plane.
- The update rule for a discretized version of the existing algorithm depends on the chosen way of conversion from the analog to the digital domain (see (3.2) and (3.3)).

Chapter 4

Experimental data

Random signals can be characterized by their probability distribution and power spectral density functions. The input signals r and b introduced in the Chapter 2 are assumed to be a random signals. So it is convenient to take a finite-length of sample data $r[k]$ and $b[k]$ and estimate their probability density function and other essential parameters for our analysis. The number of samples taken for both signals is 100,200 at the sampling rate of D= 0.5ms. Also included is an estimate of the functional relation of the input signals.

4.1 Time domain analysis

In Fig. 4.1 we have plotted some data $b[k]$ versus $r[k]$ for $k \in [50000, 51000]$. We observe the following:
- the signal b has a much smaller power (i.e., $\alpha_0 > 1$);
- roughly speaking there is indeed an ellipsoidal distribution of the samples (see Chapter 2);
- the main deviations are in the directions of b, i.e., it appears that there is (relatively) more noise in the blue signal than in the red signal.

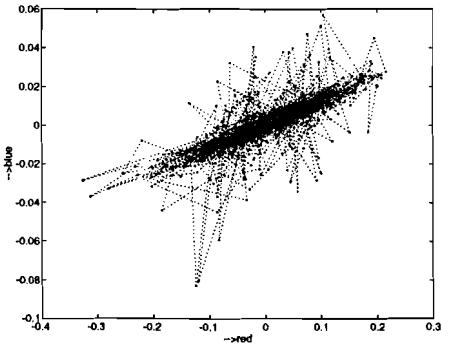

Figure 4.1: Part of the experimental data blue versus red.

In Fig. 4.2 we have plotted the histogram of the red signal. Furthermore, we have calculated the standard deviation and plotted the Gaussian probability density function given

by this standard deviation as well. We observe that histogram is in nice agreement with a Gaussian distribution. We have plotted the tails of this distribution in Fig. 4.3. We see that the distribution is slightly skew, for which the dark grains in the yarn may be responsible.

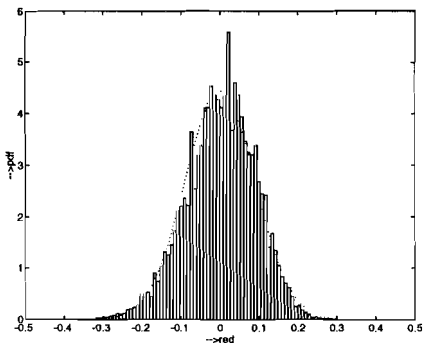

Figure 4.2: Histogram of the red signal, together with a Gaussian distribution.

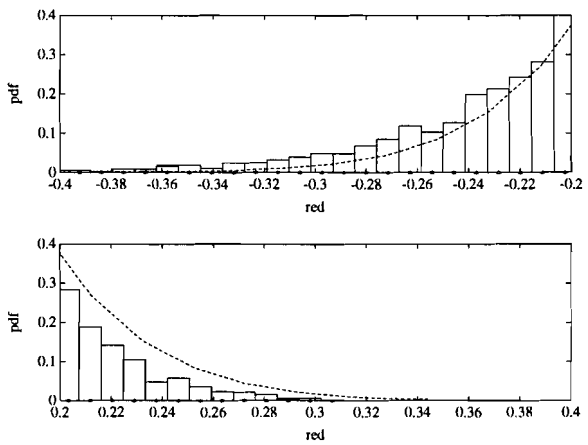

Figure 4.3: Tails of the histogram of the red signal, together with a Gaussian distribution.

In Fig. 4.4 we have plotted the histogram of the blue signal. Again, we have calculated the standard deviation and plotted the Gaussian probability density function given by this standard deviation as well. We observe that histogram does not agree with a Gaussian distribution. We have plotted the tails of this distribution in Fig. 4.5. We clearly see that there is a heavy tail in this distribution. This in agreement with our remarks on Fig. 4.1. Presumably, the histogram can be better represented as the combination of a Gaussian distribution with some other heavy-tailed distribution.

We introduce the estimators

$$\alpha_{opt} = \frac{E\{br\}}{E\{b^2\}},$$

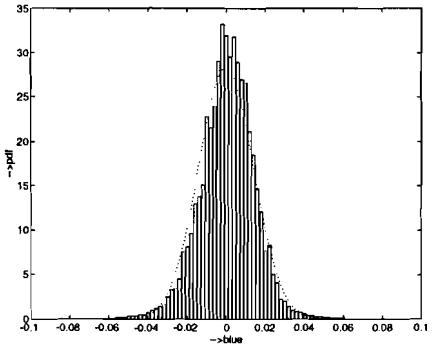

Figure 4.4: Histogram of the blue signal, together with a Gaussian distribution.

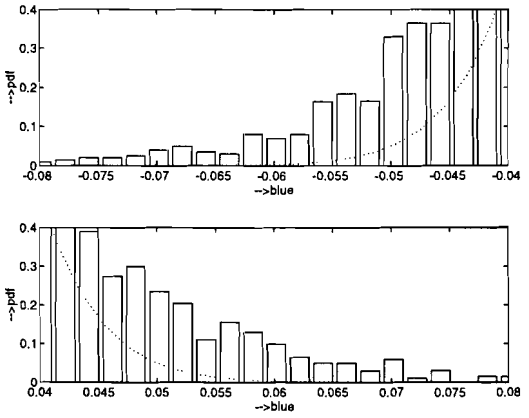

Figure 4.5: Tails of the histogram of the blue signal, together with a Gaussian distribution.

$$\beta_{opt} = \frac{E\{br\}}{E\{r^2\}}.$$

In the case of noise we get

$$\alpha_{opt} = \frac{E\{b_0 r_0\} + E\{s_1 s_2\}}{E\{b_0^2\} + E\{n_b^2\}} = \alpha_0 \frac{E\{b_0^2\}}{E\{b_0^2\} + E\{n_b^2\}} \leq \alpha_0,$$

$$\beta_{opt} = \frac{E\{b_0 r_0\} + E\{s_1 s_2\}}{E\{r_0^2\} + E\{n_r^2\}} = \beta_0 \frac{E\{r_0^2\}}{E\{r_0^2\} + E\{n_r^2\}} \leq \beta_0,$$

under the assumption that $E\{s_1 s_2\} = 0$. In other words, if we want to obtain a good estimate of α_0 (or β_0), we have to take the cleanest signal as the independent variable in the regression. It easily follows that

$$\alpha_{opt} \leq \alpha_0 \leq 1/\beta_{opt}, \tag{4.1}$$

$$\beta_{opt} \leq \beta_0 \leq 1/\alpha_{opt}, \tag{4.2}$$

and

$$\frac{E\{b_0^2\}}{E\{b^2\}}\frac{E\{r_0^2\}}{E\{r^2\}} = \alpha_{opt}\beta_{opt}. \tag{4.3}$$

In other words we have a bound for how clean the signals are by

$$\alpha_{opt}\beta_{opt} \leq \frac{E\{b_0^2\}}{E\{b^2\}} \leq 1 \tag{4.4}$$

(and an identical expression for the red signal).

In Fig. 4.6 we have plotted the standard deviations of e_1 and e_2 with

$$\begin{aligned} e_1 &= r[k] - \alpha b[k], \\ e_2 &= b[k] - \beta r[k] \end{aligned}$$

for different values of α and β. It is found that $\alpha_{opt} \approx 5.1$ and $\beta_{opt} \approx 0.125$ implying that indeed noise comes into play ($\alpha_{opt}\beta_{opt} \approx 0.64$).

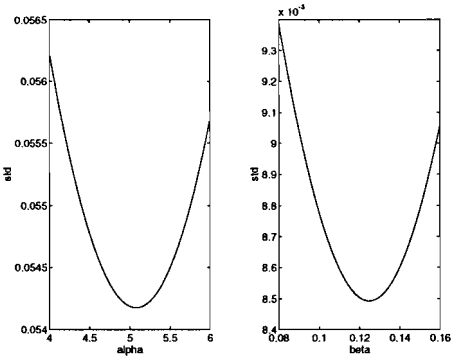

Figure 4.6: Standard deviation of the residual signals e_1 and e_2 as a function of α and β, respectively.

In Fig. 4.7 we have plotted the same data as in Fig. 4.1, but now with the lines $r = \alpha_{opt}b$ and $b = \beta_{opt}r$. It can be seen that the latter relation more closely follows the main axis from the ellipse in agreement with our earlier observation that the larger deviation from the ellipse are mostly in the blue-direction. The conclusion is that

$$\frac{E\{n_r^2\}}{E\{r^2\}} < \frac{E\{n_b^2\}}{E\{b^2\}} \tag{4.5}$$

and thus that if we want to establish a good estimate for α_0 we can better consider

$$e_2 = b - \beta r \tag{4.6}$$

and look for the minimum in power of e_2 than use e_1. Furthermore we have

$$\frac{E\{b_0^2\}}{E\{b^2\}}\frac{E\{r_0^2\}}{E\{r^2\}} = 0.64 \tag{4.7}$$

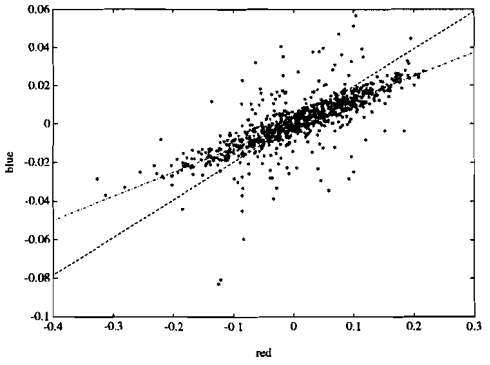

Figure 4.7: Part of the experimental data blue versus red together with the estimated functional relation according to α_{opt} (dashed line) and β_{opt} (dashed-dotted line).

indicating that at least one of the signals is seriously corrupted by noise. As bounds we have

$$0.64 \leq \frac{E\{b_0^2\}}{E\{b^2\}} \leq 1 \tag{4.8}$$

and the same is true for the red signal. From our visual inspection we infer that the red signal is cleaner than the blue one and thus

$$\begin{aligned} 0.64 &\leq \frac{E\{b_0^2\}}{E\{b^2\}} \leq 0.8, \\ 0.8 &\leq \frac{E\{r_0^2\}}{E\{r^2\}} \leq 1. \end{aligned}$$

4.2 Frequency domain analysis

In Fig. 4.8 the power spectral density functions of r and b are plotted. We observe that both spectra are different implying that either the functional relation between r_0 and b_0 is not the memoryless operator proposed previously, or that there is noise interfering.

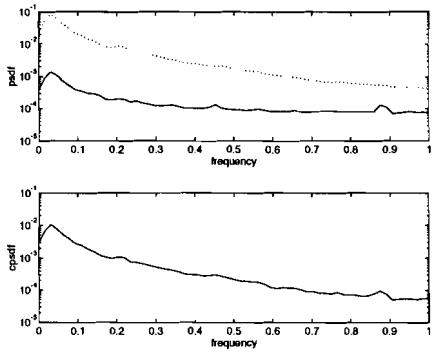

Figure 4.8: Power spectral density functions. Top figure: P_{rr} solid line and P_{bb} dashed line. Bottom figure: $|P_{rb}|$.

In Fig. 4.9 the coherence function is plotted. We note that only at the very low frequencies the coherence function is approximately 1. This implies that there is no noise interference at the low frequencies.

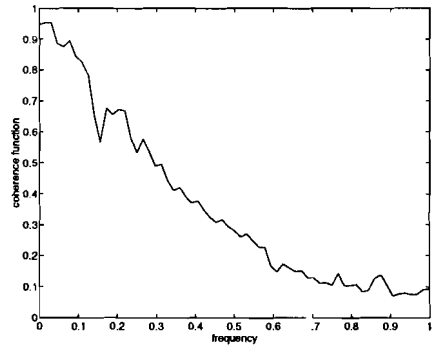

Figure 4.9: Coherence function.

In Fig. 4.10 we have plotted the transfer function estimates T_{rb} (dashed line) and T_{br}^{-1}. We note that T_{rb} is in accordance with the assumption that there is a memoryless operator relating r_0 and b_0 and that there is only a minor noise component in b. Subsequently, the fact that $T_{br} \neq 1/T_{rb}$ is attributed to a large disturbance on the blue signal. These disturbances cover a large part of the frequency spectrum and this is in accordance with the assumption of large sudden peaks in the blue signal but also reveal that these peaks occur quite frequently

(see also Fig. 4.1).

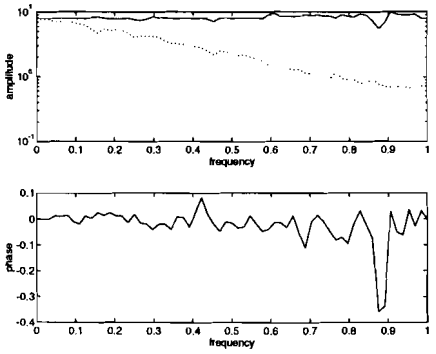

Figure 4.10: Transfer functions T_{rb} (solid line) and $1/|T_{br}|$ (dashed line). Bottom figure gives the phase of T_{rb} in radians.

4.3 Conclusions

- The data do not reject the assumption of the proposed underlying model.
- Under the assumption of the signal model, the blue signal has a relatively high noise component.
- It is better to take the red as the independent variable in the regression than the blue signal for the given data.

4.4 The error signal

Consider the error signals e_1 and e_2 defined as

$$\begin{aligned} e_1 &= r - \alpha_{opt} b, \\ e_2 &= b - \beta_{opt} r. \end{aligned}$$

We have plotted the histograms in Figs. 4.11 and 4.12 for e_1 and Figs. 4.13 and 4.14 for e_2 together with the Gaussian distribution with $\sigma_i^2 = \overline{e_i^2}$. We note that these errors do not follow a Gaussian distribution, and, furthermore, that there is a much clearer separation between a sharp Gaussian-like noise distribution and a heavy-tailed distribution for the e_2 signal. Again, this argues for e_2 as the signal in a subsequent detection stage where outliers have to be sorted out.

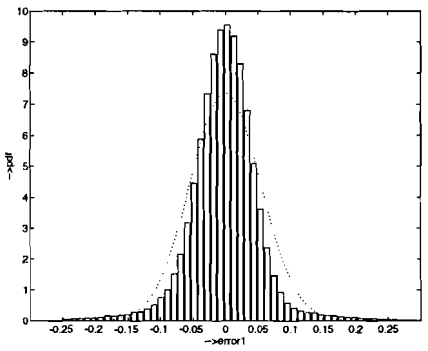

Figure 4.11: Histogram of the signal e_1, together with a Gaussian distribution.

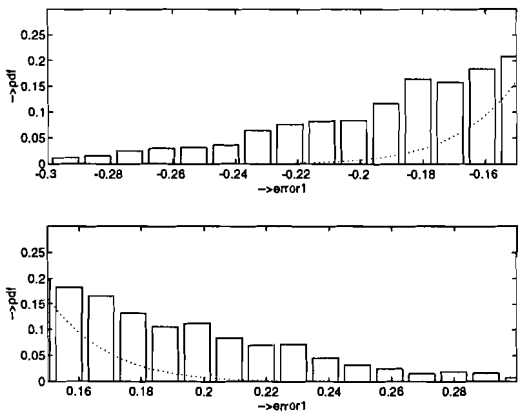

Figure 4.12: Tails of the histogram of the signal e_1, together with a Gaussian distribution.

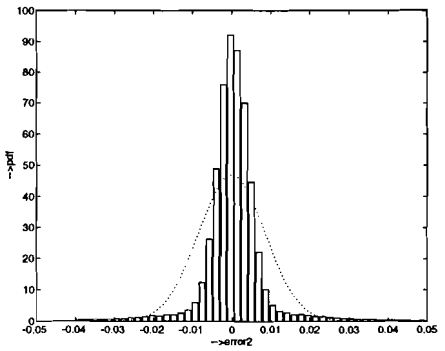

Figure 4.13: Histogram of the signal e_2, together with a Gaussian distribution.

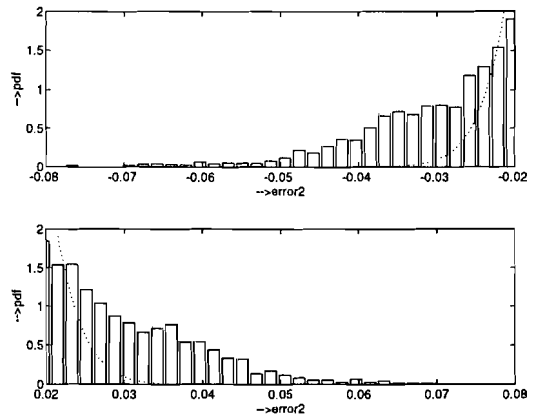

Figure 4.14: Tails of the histogram of the signal e_2, together with a Gaussian distribution.

In Fig. 4.15 we have plotted the power spectral densities of e_1 and e_2. We observe that the spectrum of e_2 is nearly flat, whereas the spectrum of e_1 peaks at the lowest frequencies. Furthermore, the spectra are nearly identical except for the lowest frequencies. It is clear that e_1 is not able to discard the low-frequent correlation that is present in the red-blue data even though this has a high degree of coherence (see Fig. 4.9). The autocorrelation function of e_1 and e_2 are plotted in Fig. 4.16 and yield the same information as the power spectral density functions.

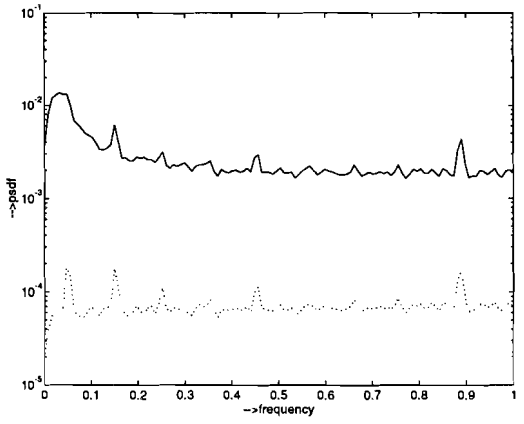

Figure 4.15: Power spectral density functions of the error signals e_1 (solid line) and e_2 (dotted line).

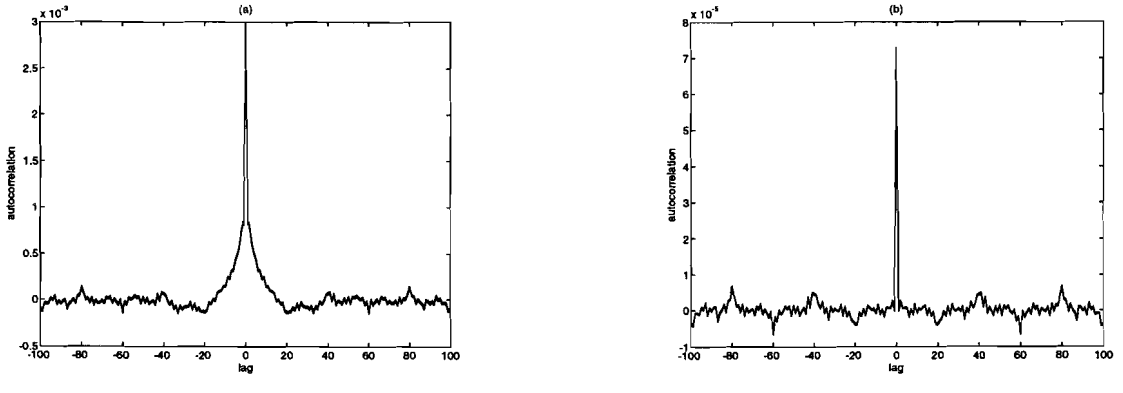

Figure 4.16: Autocorrelation function of the error signals e_1 (a) and e_2 (b).

Chapter 5

Adaptive filters

This chapter discusses the LMS and RLS algorithms. First we consider Wiener filtering and then we develop the above mentioned algorithms.

Given two variables x and d (input and reference signal, respectively), it is assumed that the functional behavior between these can be described by

$$d = w_0 x + n$$

where n is a random variable (noise) uncorrelated with d and x.

Taking the optimization criterion

$$J_{min} = \min_w E\{e^2\} = \min_w E\{(d - wx)^2\}$$

this minimal value is reached for

$$w = E\{xd\}/E\{x^2\}.$$

In adaptive filtering this is called the Wiener solution. Furthermore we have $w = w_0$. Note that this is not in agreement with our signal assumption (Chapter 2) since here it is assumed that there is no noise on the independent variable x.

5.1 The LMS algorithm

We return to the simple model

$$d = w_0 x + n.$$

The minimization criterion is taken as

$$J_{min} = \min_w E\{e^2\} = \min_w E\{(d - wx)^2\}.$$

The gradient dJ/dw is given by $-2E\{ex\}$. The stochastic gradient search is now given by updating the w in the direction of the negative gradient:

$$w[k+1] = w[k] - \frac{\mu}{2}\frac{dJ}{dw}$$

where μ is the step size. Since we do not have the disposal over dJ/dw we approximate this by its instantaneous value $dJ/dw \approx -2e[k]x[k]$ thus arriving at

$$w[k+1] = w[k] + \mu e[k]x[k].$$

The analysis of the LMS behavior is given in Appendix A, we here merely give the results. Our analysis extends the usual results on LMS adaptive behavior (e.g. [1]) to the case of non-white input and/or reference signals and is in accordance with the theory presented in [2].

Convergence speed.
With $v[k] = w[k] - w_0$ we have as initial behavior

$$E\{v[k]\} \simeq -w_0 p^k, \tag{5.1}$$

where $p = 1 - \mu\sigma_x^2$.

Weight-error fluctuations.
Taking $d[k] = w_0 x[k] + n[k]$ we have in the steady state ($k \to \infty$)

$$E\{v^2[k]\} \simeq \frac{\mu\sigma_n^2}{2} \sum_l \rho_{xx}[l]\rho_{nn}[l]. \tag{5.2}$$

with $\rho_{xx}[l]$ and $\rho_{nn}[l]$ the normalized autocorrelation function of x and n, respectively.

Error signal.
In the steady state we have

$$E\{e^2[k]\} = \sigma_n^2\{1 + \mu\sigma_x^2(1 - \frac{1}{2}\sum_l \rho_{xx}[l]\rho_{nn}[l])\}. \tag{5.3}$$

Final misadjustment.
The final misadjustment $\bar{J}$ is defined as $E\{e^2(k)\} - \sigma_n^2$ for $k \to \infty$ and from the previous expression we have

$$\bar{J} = \mu\sigma_x^2(1 - \frac{1}{2}\sum_l \rho_{xx}[l]\rho_{nn}[l]).$$

We note that for the special cases of a white noise input or a white reference signal these results are in accordance with those in [1].

5.2 The RLS algorithm

Again consider the simple model $d = w_0 x + n$. We try to estimate $E\{xd\}$ and $E\{x^2\}$ by time-averaging via the simplest mechanism. The simplest mechanism (in terms of required operations and memory) is given by

$$\begin{aligned} \hat{P}_{xd}[k] &= (1-\theta)x[k]d[k] + \theta\hat{P}_{xx}[k-1] \\ \hat{P}_{xx}[k] &= (1-\theta)x^2[k] + \theta\hat{P}_{xx}[k-1], \end{aligned}$$

i.e., a first-order recursive filter to do the time-averaging. Note that for a stationary process we have $E\{\hat{P}_{xd}[k]\} = E\{xd\}$.
Note 1. Often the RLS algorithm is developed starting from a deterministic error criterion.
Note 2. Scaling $\hat{P}_{xd}$ can avoid the term $(1-\theta)$. Doing so for $\hat{P}_{xx}$ as well, α can still be determined by the ratio.

The analysis is given in Appendix C and closely resembles that of the LMS algorithm given in Appendix A. As far as we know, the RLS algorithm has not been analyzed in this way in any of the textbooks on adaptive filtering.

Convergence speed.
For initial behavior starting with $\hat{P}_{xx}[-1] = \hat{P}_{xd}[-1] = 0$ we have

$$E\{v[k]\} = 0 \tag{5.4}$$

but $v[k]$ will initially be very noisy.
Jumping from one steady state to another, i.e, $w_0[k] = w_1$ for $k < N$ and $w_0[k] = w_2$ for $k \geq N$ we have

$$E\{w[N+k]\} = w_2 + (w_1 - w_2)\theta^k. \tag{5.5}$$

Weight-error fluctuations.
In the steady state, i.e., for $k \to \infty$, we have

$$E\{v^2[k]\} = \frac{(1-\theta)}{1+\theta}\sigma_n^2/\sigma_x^2 \sum_l \rho_{xx}[l]\rho_{nn}[l]. \tag{5.6}$$

with ρ defined as before.

Error signal.
In the steady state we find for the mean-squared error

$$E\{e^2[k]\} = \theta\sigma_n^2(1 + \frac{\theta - 1}{1+\theta}\sum_l \rho_{xx}[l]\rho_{nn}[l]) \tag{5.7}$$

Comparing (5.1)-(5.2) with (5.5)-(5.6) we see that we have a similar behavior of the LMS and RLS algorithm if $1 - \theta = \mu\sigma_x^2$ and $1 - \theta \ll 1$.

5.3 Adaptation to more robust forms

If the assumption on which the adaptive algorithms is based, i.e., $d = w_0 x + n$ is not satisfied, then clearly, such scheme can not be used. But also, if in the mean, that is for most data points d, x this signal generation model holds, but occasionally not, we may expect that performance is hampered. In that case one can resort to checking on outliers and reject these data as input data for the control loop. This notion gives rise to the following strategy:
1. determine if a data point is an outlier;
2. reject these data for an update of w.

A simple method to detect outliers is the following. We determine the variance of the error signal. If the current error signal is larger in absolute value some factor times the variance of the error signal, it is assumed to be an outlier.

Both the LMS and RLS method can be simple adjusted in the above mentioned way. In the next sections, we will propose these adapted LMS and RLS algorithms. An analysis of the behavior of these algorithms is beyond the scope of the current report. However, we can qualitatively give the expected behavior of these adaptations in comparison with the unadapted schemes (based upon the analysis there) and we will show by experiments the effects of the proposed adaptations (Chapter 6).

5.4 The adapted LMS algorithm

The LMS algorithm reads

$$\begin{aligned} e[k] &= d[k] - w[k]x[k] \\ w[k+1] &= w[k] + \mu e[k]x[k]. \end{aligned}$$

If an outlier occurs, $e[k]$ will be large. This introduces an unwanted and large step in w, from which the adaptive mechanism subsequently has to recover.

A way to prevent this is by testing the hypothesis whether a large error $e[k]$ is attributable to an outlier. To do so we can estimate the power of e (denoted by P_{ee}) and see if the current error $e[k]$ is outside a certain confidence region (say within the 1% tail of the assumed Gaussian distribution). The flow-chart of this adapted LMS algorithm is shown in Fig. 5.1.

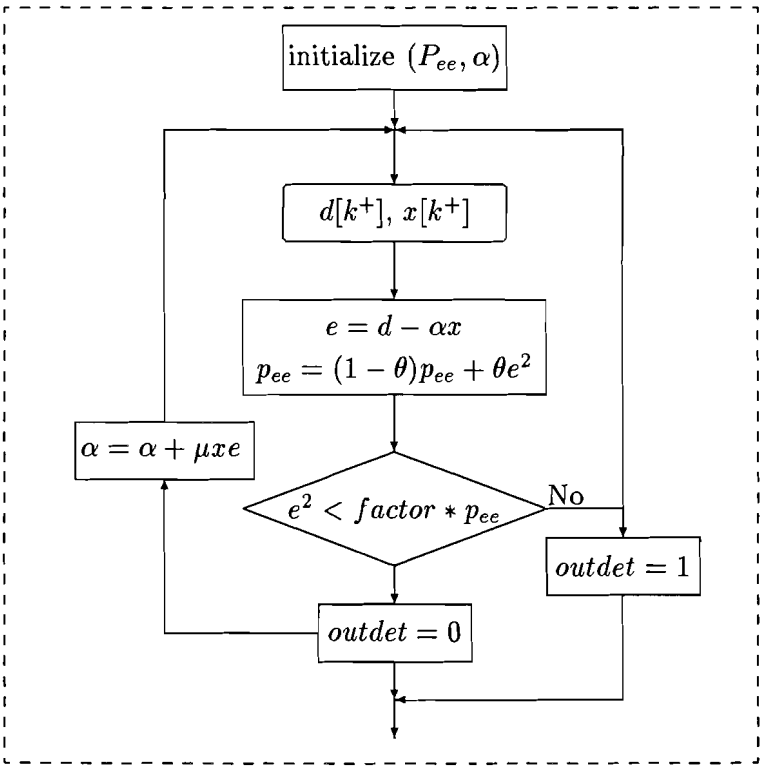

Figure 5.1: Flow-chart of the adapted LMS algorithm.

The required extra processing is minimal:

$$\begin{aligned} e[k] &= d[k] - w[k]x[k] \\ P_{ee}[k+1] &= (1-\theta)e^2[k] + \theta P_{ee}[k] \end{aligned}$$

$$\begin{aligned} \text{if} \quad & e^2[k] < factor * Pee[k] \\ & w[k+1] = w[k] + \mu e[k]x[k] \end{aligned}$$

$outdet[k] = 0$

else

$w[k+1] = w[k]$

$outdet[k] = 1$

end

A new parameter has been introduced (θ) for the estimation of the error signal power and a flag (*outdet*) has been set to indicate the detection of an outlier.

5.5 The adapted RLS algorithm

The RLS algorithm reads

$$\begin{aligned} P_{xd}[k] &= (1-\theta)x[k]d[k] + \theta P_{xd}[k-1] \\ P_{xx}[k] &= (1-\theta)x^2[k] + \theta P_{xx}[k-1] \\ w[k] &= P_{xd}/P_{xx}[k] \\ e[k] &= d[k] - w[k]x[k] \end{aligned}$$

An adaptation in analogy to the LMS-variant reads

$e[k] = d[k] - w[k-1]x[k]$ % a priori error

if $e^2[k] < factor * P_{ee}[k-1]$

$P_{dx}[k] = (1-\theta)d[k]x[k] + \theta P_{dx}[k-1]$

$P_{xx}[k] = (1-\theta)x^2[k] + \theta P_{xx}[k-1]$

$w[k] = P_{dx}[k]/P_{xx}[k]$

$outdet[k] = 0$

else % do nothing

$P_{dx}[k] = P_{dx}[k-1]$

$P_{xx}[k] = P_{xx}[k-1]$

$w[k] = w[k-1]$

$outdet[k] = 1$

end

$$\begin{aligned} e[k] &= d[k] - w[k]x[k] \quad \% \text{ a posteriori error} \\ P_{ee}[k] &= (1-\theta)e^2[k] + \theta P_{ee}[k-1] \end{aligned}$$

5.6 Comments on the adapted algorithms

- If the model holds and we use the adapted algorithms then
  - the weight will converge to the Wiener solution w_0;
  - the weight has less fluctuations than in the nonadapted case;

- the excess-mean-squared error will be less, but this will probably be hardly noticeable in the mean-squared error.

- If the model does not hold because the x component contains noise thus

$$\begin{aligned} x &= b + n_2 \\ y &= r + n_1 \\ r &= \alpha_0 b \end{aligned}$$

then

- the weight will not converge to the Wiener solution ($w_0 = E\{xy\}/E\{x^2\}$) but will be closer to the 'true' value α_0;
- the weight has less fluctuations than in the usual LMS/RLS case;
- mean-squared error will be larger than in the usual LMS/RLS case since the algorithm does not converge to the Wiener solution.

Furthermore it is noted that we need to consider the initialization of these adapted algorithms carefully since we need a good initial estimate of P_{ee} for a proper functioning.

Chapter 6

Experiments with different algorithms

In this chapter we present computer simulation of the algorithms discussed in chapter 5. The programs were developed in MatLab.

The choice of one algorithm in favor of another is determined by various factors, depending on the exchange between complexity requirements and convergence properties of the adaptive filter. In this section the main concern will be on the convergence properties of the adaptive filters.

6.1 Filtered error signal

In this section we repeat the experiments of Chapter 4, but now with the filtered e_1 and e_2 (filtered by FIR filter [1/2, 1/2]). The motive is to show the effect of filtering on the error signal as is suggested by the difference between the discretized versions of the existing adaptive system, see (3.3) and (3.2).

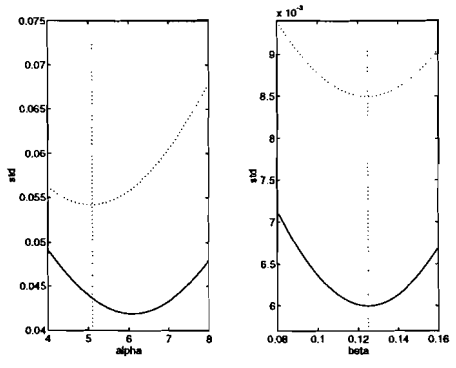

Figure 6.1: Standard deviation of the filtered (solid line) and unfiltered (dashed line) signals e_1 (left) and e_2 (right).

In Fig. 6.1 we have plotted the standard deviation of the filtered and unfiltered signals e_1

and e_2 for different values of α and β.

This plot shows that:

- α_{opt} is shifted from 5.1 to approximately 6.1 by the introduction of the filtering;

- β_{opt} remains the same for the filtered and unfiltered signal e_2.

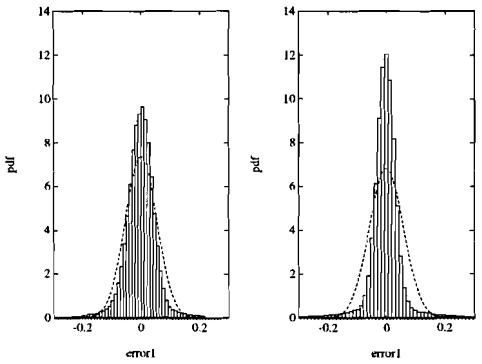

Figure 6.2: Histogram of the unfiltered signal e_1 (left figure) and the filtered signal e_1 (right figure), together with a Gaussian distribution.

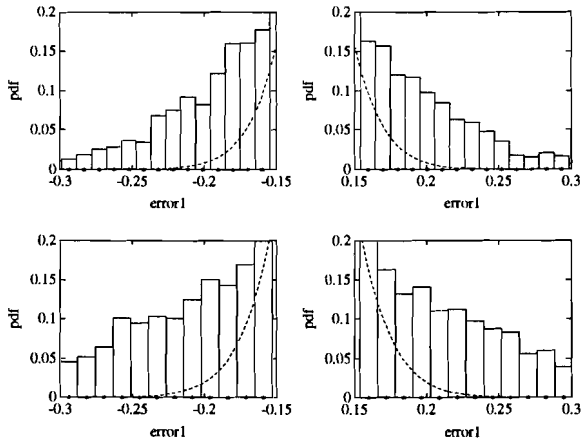

Figure 6.3: Tails of the histogram of the signal e_1 using the normal LMS algorithm (top figure) and using an update rule by first filtering e_1 (bottom figure), together with their Gaussian distribution.

In Fig. 6.2 is plotted the histograms together with the Gaussian distribution of the using the normal LMS algorithm and the one with filtering of e_1. The tails of this distribution are

plotted in Fig. 6.3.

These plots shows that:

- There is a heavy tail in the distribution introduced by the filtering of e_1;

- The distribution of the filtered and unfiltered e_2 is almost the same.

Conclusions

- The adaptive weight α is very sensitive to filtering e_1.
- The adaptive weight β is not sensitive to filtering e_2.
- Using the error signal $e_2 = b - \beta r$ gives the same result with different rules of discretization.

6.2 Adaptive algorithms

Experiment 1: existing adaptive mechanism.

We simulate first the existing adaptive mechanism by its discretized version using the 'trapezium rule' as it is described in Chapter 3. The simulation algorithm is that given by (3.2). The resulting behavior of $\alpha[k]$ and the optimal solutions ($\alpha_{opt} = 5.1; 1/\beta_{opt} = 8$), which are found from the experimental data in Section 4.1, are plotted in Fig. 6.4.

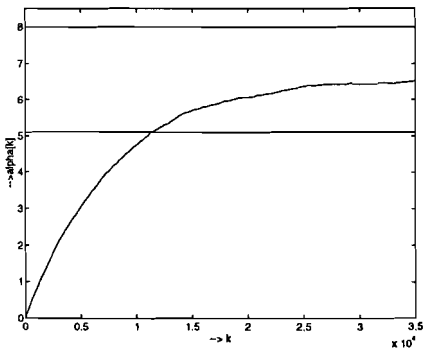

Figure 6.4: The behavior of $\alpha[k]$ with existing adaptive mechanism, together with α_{opt}=5.1 and $1/\beta_{opt}$ =0.125.

The result of this simulation shows the following.

- When the number of iterations k approaches infinity, α converges to an unknown optimum solution. This solution is solution within the bounds given in Section 4.1.

- Almost a smooth adaptation mechanism, which means a small ($E\{v^2[k]\}$).

- Slow convergence rate.

- The adaptation weight α converge to the optimum solution within the bounds given in section 4.1.

Experiment 2: LMS algorithm for α.

For this experiment $\alpha[k+1] = \alpha[k] + \mu b[k]e_1[k]$ is used to update the adaptive weight $\alpha[k]$. The step size μ is set to 1.025, in order to get approximately the same performances to the existing adaptive mechanism according to (3.5). The resulting behavior of $\alpha[k]$ is plotted in Fig. 6.5.

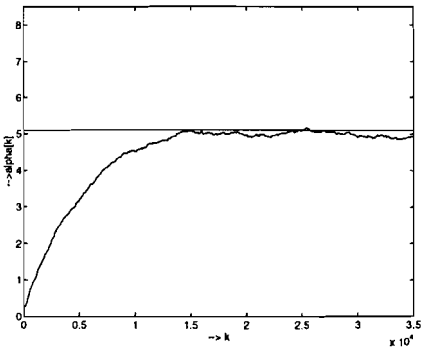

Figure 6.5: The behavior of $\alpha[k]$ with LMS algorithm.

The result of this simulation shows that:

- The adaptive weight α does converge to an optimal value and after convergence, $\alpha[k]$ continues to fluctuate around this optimal value.

- This convergence value is in agreement with the optimal value found in section 4.1 (α_{opt}).

Comparison of results Exp.1 and Exp.2.

- The rate of convergence in Exp.2 is approximately twice as fast as in Exp.1, while the variance is larger.

- The large fluctuation after convergence is caused by outliers: an outlier causes a large value for e_1, inducing a large step in α from which the algorithm has to reconverge to its optimal value.

- The LMS algorithm suffers from noise interference at the high frequencies and as a result it gives a larger final misadjustment.

Experiment 3: LMS algorithm and RLS algorithm for β.

We simulate now the LMS algorithm with the equation $\beta[k+1] = \beta[k] + \mu r[k]e_2[k]$ and the RLS algorithm with the equations shown in section 5.2. In order to get approximately the

same performances to the existing adaptive mechanism, the step size μ is set to 0.125 for LMS algorithm and the forgetting factor θ is set to 0.998 for the RLS algorithm. The resulting behavior of $\beta[k]$ with the LMS algorithm and the RLS algorithm is plotted in Fig. 6.6a and Fig. 6.6b, respectively.

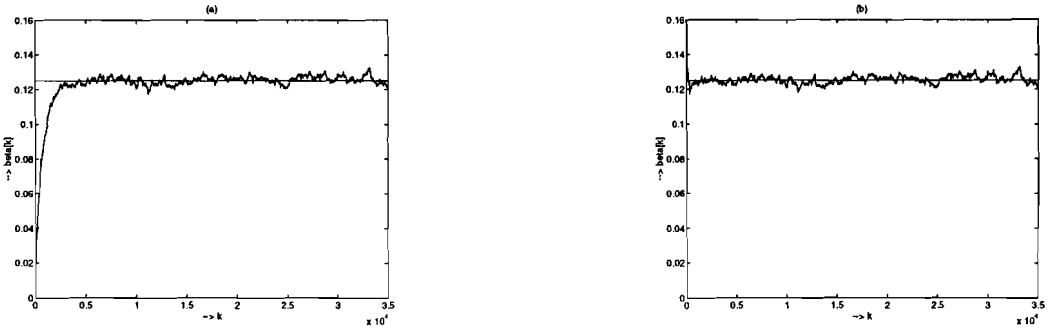

Figure 6.6: The behavior of $\beta[k]$ with LMS algorithm and RLS algorithm. (a) LMS ; (b) RLS.

The result of this simulation shows that:

- For the RLS algorithm the initial behavior is very noisy as is expected from the analysis of Section 5.2. (This is not very good visible in this plot.)

- When the number of iterations k approaches infinity, β converges to the optimum solution found in Section 4.1 (β_{opt}) for both algorithms.

- With both algorithms $\beta[k]$ fluctuate after convergence around β_{opt} in nearly the same way.

- The rate of convergence with the RLS algorithm is faster than that of the LMS algorithm.

Comparison of results Exp.3 and Exp.2.

- Exp3 shows a faster rate of convergence for both algorithms compared with Exp.2.

Experiment 4: The robust LMS algorithm for α.

We simulate here the robust LMS algorithm, explained in section 4.3 with $\theta = 0.95$ and $factor = 2$. The initialization is done by adapting the first 2000 samples with the usual LMS algorithm, in order to reach a reasonable estimate for P_{ee}. The simulation results of $\alpha[k]$ is plotted in Fig. 6.7.
The result of this simulation shows that:

- There is a smooth adaptation process, which means a small $E\{v^2[k]\}$. This is in agreement with the remarks in Section 5.6.

- α converge to the unknown optimal solution which is higher than the optimal solution given in Section 4.1 (α_{opt}).

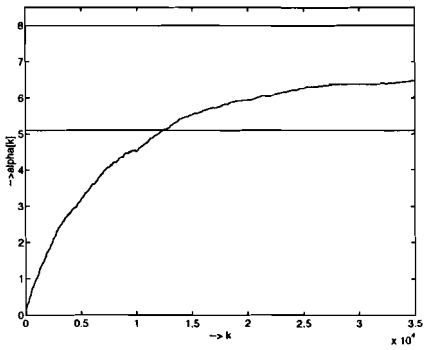

Figure 6.7: The behavior of $\alpha[k]$ with the robust LMS algorithm.

- This adaptive algorithm shows almost the same behavior as the adaptive mechanism in Exp.1 with the exception of the convergence rate, which is faster than in Exp.1.

- α shows a smooth adaptation which is a result of avoiding unwanted and large step in α as is discussed in Section 5.4.

Conclusion:
On the basis of the given data and the performed experiments we prefer the unadapted LMS algorithm with $e_2 = b - \beta r$ as error signal since it is the simplest mechanism.

Chapter 7

Implementation

The signal processing workplace consists of hardware and software for developing and implementing signal processing algorithms on sophisticated hardware and software. To make use of this sophisticated hardware, a programming environment is required which allows users to quickly develop and implement an application. Therefore, it must be possible to describe the application in some sort of high level language, which can be easily implemented on the hardware.

The software that enables us to do this consists of a real-time operating system, called Virtuoso, and some development tools (assembler, compiler and linker) from Texas Instruments. With Virtuoso and the development tools, it is possible to write an application in ANSI-C and download it on the hardware. A detailed description of the hardware and the Virtuoso real-time operating system can be found in [6].

7.1 C-program

The source file, for implementing the front-end of the digital quality surveillance system with the 'C40, is shown in Appendix D. Translating this source file into a code that the 'C40 can execute is a process consisting of several steps (see Fig. 7.1).

The cl30 shell program is a utility that can compile, assemble, and optionally link in one step. The shell runs one or more source modules through the following steps:

- The compiler which includes the parser, the optimizer and the code generator.
- The assembler which generates a COFF object file.
- The linker (optional) which links the files to create an executable object file.

The C compiler:

The TMS320 floating-point C compiler is a full-featured optimizing compiler that translates standard ANSI-C programs into TMS320C4x assembly language source. The compiler is made up of three distinct programs: the parser, optimizer, and code generator.

The input for the parser is a source file. The parser reads the source file, checking for syntax and semantic errors, and writes an internal representation of the program called an intermediate file. The optimizer is an optional pass that runs between the parser and the code generator. The input is the intermediate file produced by the parser. The input for the

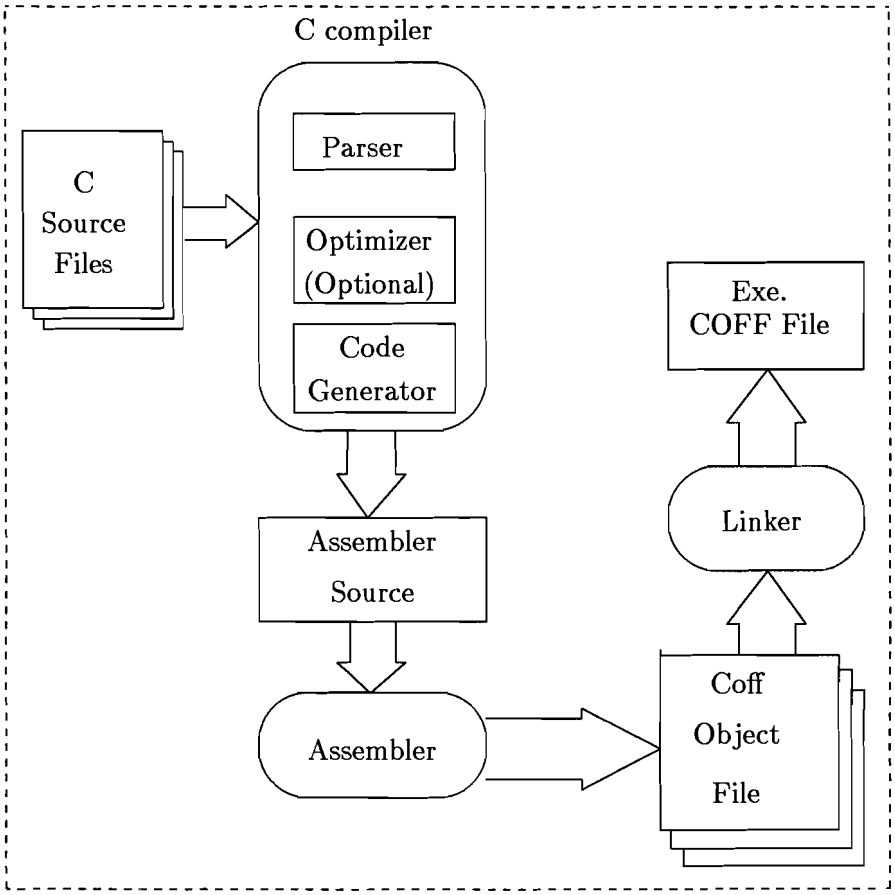

Figure 7.1: Program development flow.

generator is the intermediate file produced by the parser or the .opt file from the optimizer. The code generator produces an assembly language source file.

The assembler:

The assembler translates assembly language source files into machine language object files. Source files can contain instructions, assembler directives, and macro directives. You can use assembler directives to control various aspects of the assembly process, such as the source listing format, data alignment, and section content.

The format of these object files is called common object file format, or COFF. COFF allows you to define your systems memory map at link time. This maximizes performance by enabling you to link C code and data objects into specific memory areas.

The linker:

The linker combines object files into a single executable object module. As it creates the executable module, it performs relocation and resolves external references. The linker accepts relocatable COFF object files (created by the assembler) as input. Linker directives allow you to combine object file sections, bind sections or symbols to addresses or within memory ranges, and define or redefine global symbols.

The source file:

The C program-flow for implementing the LMS algorithm in the C40 is shown in Fig. 7.2. The program must handel the computation, as well as the input/output, between these a synchronization is required. Therefore the program-flow is divided into two blocks named read block and main block.

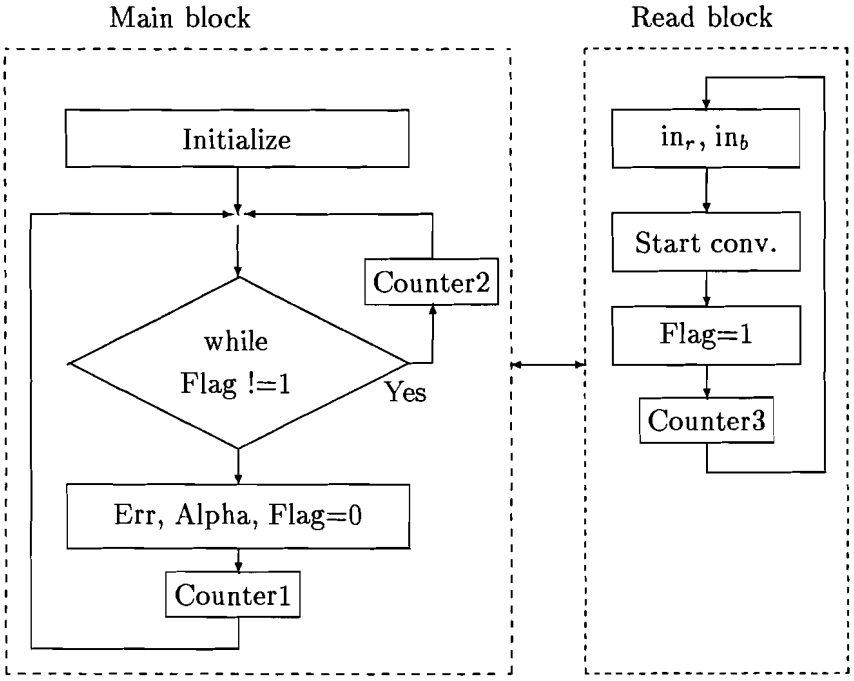

Figure 7.2: Program-flow of the implemented C program.

The read block is used for setting the flag in ready state (Flag=1) after reading and converting of a data sample is completed. This block is realized with a C function by using a special naming convention c_intxx where xx is a two-digit interrupt number between 00-99. The name c_intxx is the C entry point which is reserved for the system reset interrupt. This special interrupt routine initializes the system and calls the function main.

The main block initializes and calibrates the AD and DA converters first and then starts the LMS algorithm. After completing the LMS algorithm (calculating the error signal e and updating the adaptive weight) the error signal e is sent to the output device and the flag to the wait state (Flag=0).

These two blocks are operating in parallel. The interrupt handling routine (KS_EnableISR(IRQ, c_intxx)) install an interrupt service routine, which provides the c_intxx to operate entirely in the background. The fastest working of this program can be achieved by setting the sampling period of the AD/DA converter to a value where the main block has to wait a minimum possible time for the new sample. So it is necessary to measure the execution time of both blocks in order to reach a synchronized working of these blocks. There

are different methods to measure the execution time (see [9]). A simple method is used here to optimize the operations of these blocks. Three counters are included as shown in Fig. 7.2. By repeating the main block a specified number of time, we can observe whether the executing time of these blocks leads to a minimum waiting time. If counter1 is less than counter3, then the samples passed to the main block has a sampling period which is less than the sampling period of the AD/DA converter.

The complete C program for implementing the LMS algorithm as it is described above is shown in Appendix D. This appendix also contains the listing of a C program for saving the red signal (r), the blue signal (b) and the error signal from the analog quality surveillance system on the files named out_r, out_b and out_y, respectively.

7.2 DSP-system

This section describes the TMS320C40 Digital Signal Processor architecture. It is intended to serve as a quick look-up of the 'C40 architecture for the application programmer. Detailed information can be found in the 'TMS320C40 User's Guide" from Texas Instruments.

The 'C40's high performance is achieved through the precision and wide dynamic range of the floating-point units, large on-chip memory, a high degree of parallelism, and the six-channel DMA coprocessor. The block diagram of the 'C40 is shown in Fig. 7.3.

The interface to the outside world is done via two programmable memory ports. Two 4K bytes internal RAM blocks are available as well as a small cache of 512 Bytes. The 6 FIFO buffered communication ports providing a total peak bandwidth of up to 320MB/s and they can also directly be interfaced to peripheral devices. Internally there is a six-channel DMA coprocessor that permits to execute memory operations while the CPU is handling computational tasks.

The Central Processing Unit (CPU)

The 'C40 has a register-based CPU architecture. The CPU comprises the following components:

- Floating-point / integer multiplier
- ALU for performing arithmetic: floating-point, integer, and logical operations
- 32-bit barrel shifter
- Internal buses (CPU1/CPU2 and REG1/REG2)
- Auxiliary Register Arithmetic Units (ARAUs)
- CPU register file

The most important ones for the application programmer are the CPU registers. The 'C40 primary register file provides 32 registers in a multiport register file that is tightly coupled to the CPU. All of the primary register file can be operated upon by the multiplier and ALU, and can be used as general-purpose registers. However, the registers also have some special functions. Besides the CPU primary register file the expansion register file contains two special registers that act as pointers. We refer to [8] for detailed information on the CPU

registers and various CPU components.

Memory Organization

The total memory reach of the 'C40 is 4G (giga) 32-bit words (16 Gbytes). Program memory (on-chip RAM or ROM and external memory) as well as registers affecting timers, communication ports, and DMA channels are contained within this space. This allows tables, coefficients, program code, and data to be stored in either RAM or ROM.

RAM block 0 and block 1 are 4K bytes each. The ROM block is reserved and contains a boot loader. Each RAM and ROM block is capable of supporting two accesses in a single cycle. The separate program buses, data buses, and DMA buses allow for parallel program fetches, data reads and writes, and DMA operations. For example: the CPU can access two data values in one RAM block and perform an external program fetch in parallel with the DMA coprocessor loading another RAM block, all within a single cycle.

Peripherals

All 'C40 peripherals are controlled through memory-mapped registers on a dedicated peripheral bus. This peripheral bus permits straightforward communication to the peripherals. The 'C40 peripherals include two timers and two serial ports.

The six high-speed communication ports provide rapid processor-to-processor communication through each port's dedicated communication interfaces.

Communication port features:

- 160-megabit per second bidirectional data transfer operations (at 40-ns cycle time)
- direct processor-to-processor communication via eight data lines and four control lines
- buffering of all data transfers, both input and output
- automatic arbitration provided to ensure communication synchronization
- synchronization between the CPU or direct-memory access (DMA) coprocessor and the six communication ports via internal interrupts and internal ready signals

The six channels of the on-chip Direct Memory Access (DMA) coprocessor can read from or write to any location in the memory map without interfering with the operation of the CPU. The two timer modules are general-purpose 32-bit timer/event counters with two signaling modes and internal or external clocking. They can signal internally to the 'C40 or externally to the outside world at specified intervals, or they can count external events.

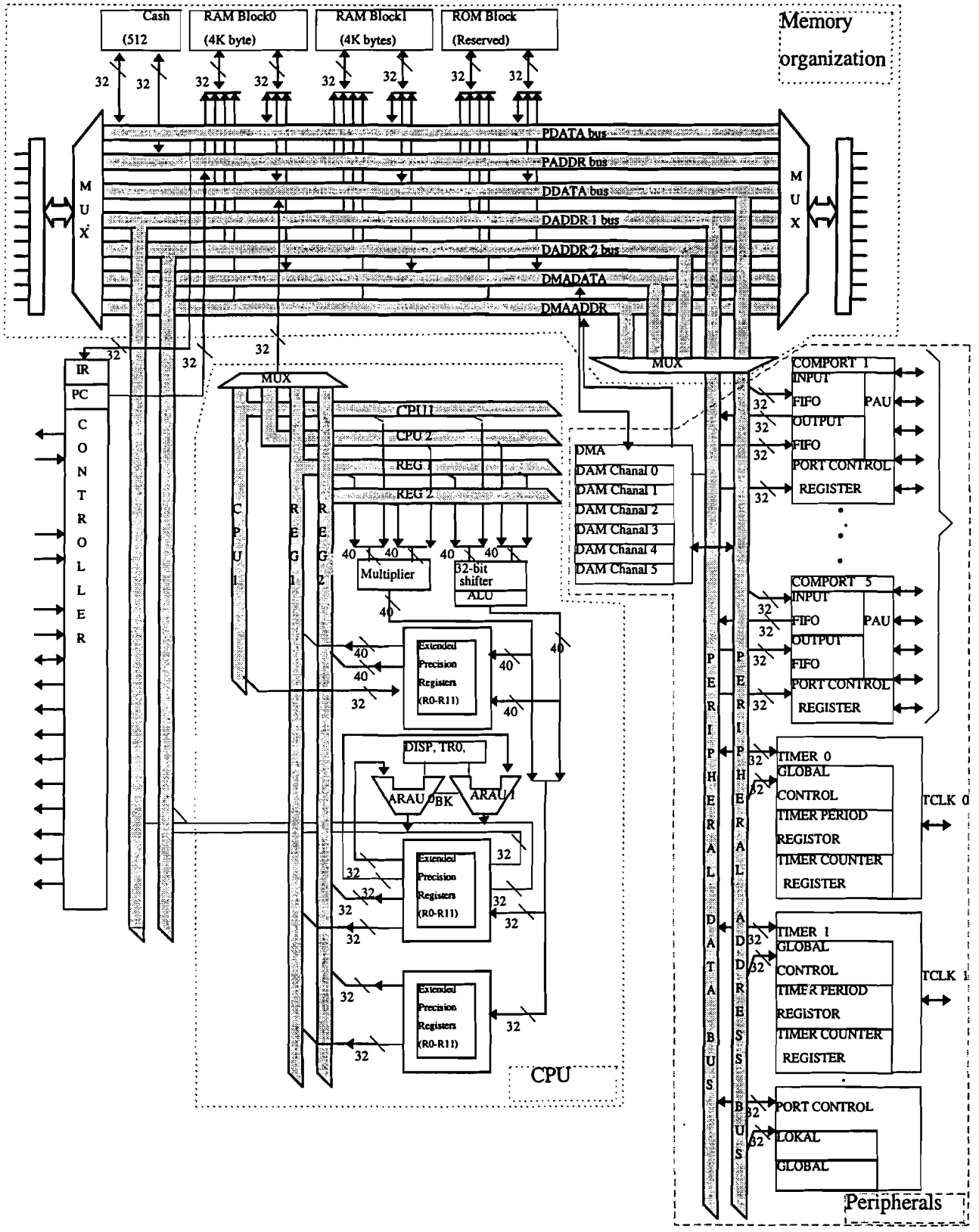

Figure 7.3: TMS320C40 block diagram.

Chapter 8

Evaluation

This chapter presents the experimental set-up of the digital quality surveillance system. The test results of the experimental set-up and the possibility of minimizing the costs of the DSP system are also discussed.

8.1 Experimental set-up

In Fig. 8.1 is shown the experimental set-up. The mechanical system for rotating the thread is denoted by M. The sensor S produces the illuminating light I and measures the reflected light R. The sensor produces the signals r and b which are amplified by A_1 and A_2 and fed to the A/D converters. These signals are input to the DSP system. The output of the DSP system is the error signal e which goes through a D/A converter and is made visible on the oscilloscope (Osc) together with the error signal y of the existing analogue system. The DSP system can be controlled by the PC.

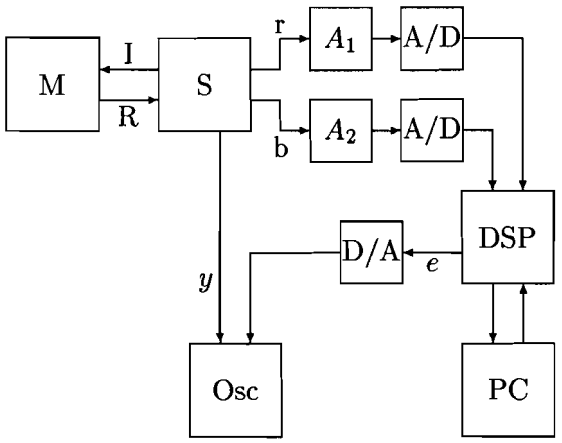

Figure 8.1: Experimental set-up.

The sensor (S) and the mechanical system (M)

A ± 3 m string of thread containing a ± 3 cm green colored part was placed over 2 wheels with circumference of 50 cm. One of the wheels was driven by a motor with adjustable speed.

This string of thread was lead through the sensor (see Fig. 1.1) producing the r and b signals.

The two analog amplifiers (A1, A2)
These analog amplifiers were used to obtain a sufficient dynamical range for the A/D conversion.

The A/D converters (A/D)
The A/D converters are part of the Signal Processing Workplace. These converters have a 12 bit resolution with a dynamical range of ± 10 V and they are designed to perform synchronous analog to digital conversion of the two analog inputs. These two channels offer high conversion speed with a sharp Butterworth anti-aliasing filter (LPF 100kHz). The sampling frequency is set to 1kHz, which can be adjusted in software (see [10]).

The TMS320C40 floating point DSP (DSP)
The DSP system is used to calculate the error signal e and the adaptive weight α. All variables are presented on 40-bit floating-point values. The 'C40 implementation of floating-point arithmetic allows for floating-point operation at fixed-point speeds via a 40-ns instruction cycle. The output of the DSP is the error signal e.

The PC (PC)
The TMS320C40 uses a fully implemented 'Kernighan and Ritchie' C compiler for converting a C language program (our program is shown in Appendix D) into a TMS320 assembly language program. The TMS320 Assembler/Linker translates TMS320 assembly language source code into an executable object model. The PC downloads the output of this assembler/linker on the hardware (PD-TIMEX modular DSP system). The DSP communicates also with the PC using the available communication mechanism.

The D/A converter (D/A)
The D/A converter is also part of the Signal Processing Workplace. It offers high conversion speed with sharp 4th order Bessel filters (LPF 100kHz) and 16 bits precision.

8.2 Testing the system

Several test were performed in order to consider the operation of the system.

Test 1. Collecting the data
Aim of this experiment was to test the measurement system, in particular the operation of the mechanical system, sensor, analogue amplifiers, A/D converters and the DSP - hard disk communication. The data was read into the DSP and subsequently saved on the hard disk of the PC. The collected r and b data were supposed to be similar to those used in Chapter 4 and Chapter 6 (with the exception of the data representing the green piece).

Fig. 8.2 shows the collected data. This should be compared to Fig. 4.1. Apart from the amplification we note that both the ellipsoidal structure and the outliers (the green part) are clearly visible. We also note that the ellipse is much more well-defined than in the original data. This observation lead to the following test.

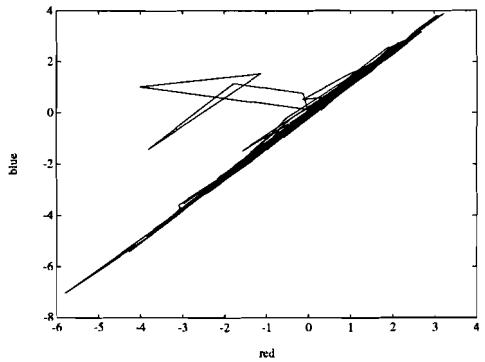

Figure 8.2: Part of the data blue versus red.

Test 2. Comparison with the original data

From the uncontaminated data, we estimated α and β as before (Chapter 4) and obtained $\alpha = 0.7998$, $\beta = 1.2354$ and $\alpha\beta = 0.988$. We infer that for this piece of thread both the red and blue signal contain only a minor noise contribution n_1 and n_2 (see Chapter 4). Therefore it raises the question how representative the original data was. On the other hand, the data we collected represents only a small piece of thread (3 m.).

Test 3. The adaptive system

We applied the LMS-algorithm to the collected data (in MatLab) with $\mu = 0.001$. Since the data contained only minor noise components n_1 and n_2 both $e_1 = r - \alpha b$ or $e_2 = b - \beta r$ could be equally well applied. We considered e_1 since this signal is most directly comparable to the y-signal of the existing system (see Chapter 3). The resulting error signal e_1 is shown in Fig. 8.3. It shows clearly the outliers (green part) at the same time-instants as the y-signal. These parts can be easily identified by using thresholding. Just before the green part the occurrence of a knot is also clearly visible.

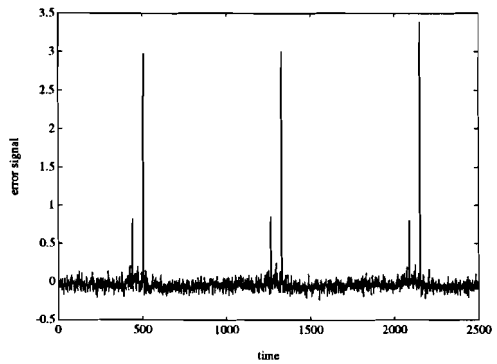

Figure 8.3: The error signal e_1.

Test 4. Signal quantization

In order to test if the system can operate with more coarsely quantized input data, the 12 bits r and b representation was reduced to an 8 bits representation. The error signal contained slightly more noise but the outliers remained clearly visible. With the 6 bits r and b representation the error signal contained a relative large amount of noise due to quantization but still the outliers remained clearly visible.

Test 5. Arithmetic accuracy

The mean value of α is roughly 1 as a consequence that both the signals r and b are scaled for the A/D conversion. The noise in $\Delta\alpha$ is approximately 0.08. This means the quantization noise should be much less than 0.08. Thus approximately 16 bits should be sufficient to calculate and update α. Therefore, a cheap fixed-point processor is possible.

Test 6. Real-time adaptive filtering

The LMS-algorithm implemented in the C-program (see Appendix D) was converted into an executable object model (Chapter 7) and downloaded on the DSP. The error signal e was compared with the analogue error signal y of the existing system on an oscilloscope (see Fig. 8.1). In both signals the occurrence of the deviations were clearly visible, which is in agreement with the MatLab tests.

The LMS algorithm can be easily replaced by the RLS algorithm or their robust variants simply by replacing the LMS algorithm part in the program listing shown in appendix D with the chosen algorithm .

8.3 Conclusions

The experimental set-up is sufficient to test the algorithms. However, it would be more convenient to be able to use larger segments of the yarn.

The signals that are collected resemble the original data. However, there seems to be less disturbances in the data collected in our experiments.

The colored parts in the thread can be easily identified by adaptive filtering and thresholding. However, tests with other disturbances are recommendable.

The real-time LMS algorithm works and can be easily adapted to an RLS adaptive mechanism.

A larger signal quantization (at least as low as 8 bits) is possible without disturbing the further processing. This means that cheap A/D converters can be used.

8.4 Recommendations

The assembler code has to be optimized if a number of parallel processors has to be dealt with. The number of parallel processes that can be run on a single DSP for a given sample frequency has as yet to be determined. This can be done as follows.

- Set the A/D converter sampling period to a maximum sampling period where the detecting of the outliers is possible.

- Measure the total execution time (for all parallel processes) of the necessary operation to be performed to implement LMS algorithm. This can be done in several methods. Here we give an example based on the timing function on 96002 from Ariel (see [11]).

```
extern void timer0_irqh;       /*interrupt routine */
KS_EnableISR (4,timer0_irqh); /* enable interrupt */

int main ()
{
  int time_1, time_2
  timer0_init();               /* initialize timer */
  time_1=time0_read();         /* read time_1      */

  ...

  time_2=time0_read();         /* read time_2      */

 /* time_2 - time_1=execution time in microseconds*/

  ...
}
```

- Adjust this above mentioned execution time, by altering the number of process, till the executing time of the reading block.

Chapter 9

Conclusions and recommendations

This thesis has lead to the following conclusions.
- The existing system is basically an LMS-type algorithm.
- Depending on which input signal is most disturbed the regression parameter should be α or β.
- As analog counterparts of the existing system an LMS or RLS algorithm can be used and with proper parameter setting these algorithms will have a quite similar performance.
- It deserves consideration to use more robust adaptive algorithms since the data is essentially not in agreement with the data model for which adaptive filters are intended.
- Software has been designed for a DSP system in order to arrive at a digital counterpart of the existing analog system.
- Our simulations suggest that cheap A/D converters and a low-cost fixed-point DSP can be used to implement the front end of the quality surveillance.

The presented work can be extended in several ways.
The system can be implemented on a fixed-point DSP. This requires considering and minimizing the effect of quantization noise.
So far the occurrence of a perturbed signal is determined by considering the error $e = b - \alpha r$ only. In essence, the shortest axis of the diagonal distribution is determined and taken as a measure to determine outliers. It is better to consider a unitary transformation on the data according to

$$\begin{pmatrix} e \\ f \end{pmatrix} = \begin{pmatrix} 1 & -\alpha \\ \alpha & 1 \end{pmatrix} \begin{pmatrix} r \\ b \end{pmatrix} \sqrt{1+\alpha^2}.$$

In that way outliers in the direction of the main axis (i.e., the signal f) of the ellipsoidal distribution can be dealt with as well.
The existing decision mechanism, presently in cascade with the analog system, is a digital system. Presumably, the front-end adaptive system and the decision making mechanism can be incorporated in a single flexible DSP system.
The existing system as well as its proposed analog counterpart essentially describe the color in a two-dimensional space. Since the human eye describes colors in a three-dimensional space, an red-green-blue detector should in principle yield a better color deviation discrimination.

Appendix A

The convergence properties of the LMS algorithm.

In this appendix the convergence properties of the LMS algorithm with one coefficient is derived. We assume here the simple model

$$d[k] = w_0 x[k] + n[k]$$

and the LMS update rule

$$w[k+1] = w[k] + \mu x[k] e[k]$$

where

$$e[k] = d[k] - w[k]x[k].$$

For this, the weight error ($v[k]$) and the error signal ($e[k]$) are represented as follows:

$$v[k] = w[k] - w_0,$$

$$e[k] = d[k] - w[k]x[k] = w_0 x[k] + n[k] - w[k]x[k] = n[k] - v[k]x[k].$$

Weight-error fluctuation.
The update equation for the weight error is:

$$v[k+1] = v[k](1 - \mu x^2[k]) + \mu x[k] n[k]. \tag{A.1}$$

For small $\mu x^2 \Rightarrow E\{\mu x^2\} = \mu \sigma_x^2 \ll 1$

$$\begin{aligned} v[k+1] &\simeq v[k](1 - \mu\sigma_x^2) + \mu x[k] n[k] \\ &= v[k]p + \mu x[k] n[k] \\ &= \mu \sum_{l=0}^{\infty} p^l x[k-l] n[k-l] \end{aligned}$$

with $p = 1 - \mu\sigma_x^2$.

$$E\{v^2[k+1]\} \simeq p^2 E\{v^2[k]\} + 2\mu p E\{v^2[k] x[k] n[k]\} + \mu^2 E\{x^2[k] n^2[k]\}$$

With $E\{v^2[k+1]\} = E\{v^2[k]\}$ and $\mu\sigma_x^2 \ll 1$ we have

$$\begin{aligned} 2\mu\sigma_x^2 E\{v^2[k]\} &= 2\mu p E\{v[k]x[k]n[k]\} + \mu^2 E\{x^2[k]n^2[k]\} \\ &= 2\mu p E\{v[k]x[k]n[k]\} + \mu^2\sigma_x^2\sigma_n^2. \end{aligned}$$

Substituting $v[k]$ in the right hand side of the previous equation results in:

$$\begin{aligned} 2\mu\sigma_x^2 E\{v^2[k]\} &= 2\mu p E\{\mu\sum_{l=0}^{\infty} p^l x[k-1-l]n[k-1-l]x[k]n[k]\} + \mu^2\sigma_x^2\sigma_n^2 \\ &= 2\mu^2\sum_{l=1}^{\infty} p^l E\{x[k]x[k-l]\}E\{n[k]n[k-l]\} + \mu^2\sigma_x^2\sigma_n^2 \\ &= 2\mu^2\sum_{l=1}^{\infty} p^l r_{xx}[l]r_{nn}[l] + \mu^2\sigma_x^2\sigma_n^2 \\ &= \mu^2\sum_{l=-\infty}^{\infty} p^{|l|} r_{xx}[l]r_{nn}[l]. \end{aligned}$$

Let L denote the effective length of $r_{xx}[l]r_{nn}[l]$ and let $1 - p^L \ll 1$ (i.e., the effective length of $p^l \gg$ effective length of $r_{xx}[l]r_{nn}[l]$). Then

$$E\{v^2[k]\} \simeq \frac{\mu}{2}\sum_{l=-\infty}^{\infty}\frac{r_{xx}[l]}{\sigma_x^2} r_{nn}[l] = \frac{\mu}{2}\sum_{l=-\infty}^{\infty}\rho_{xx}[l]r_{nn}[l]$$

with $\rho_{xx}[l] = r_{xx}/\sigma_x^2$.
So, finally we get for the relative weight error:

$$\frac{E\{v^2[k]\}}{w_{opt}^2} = \frac{\mu\sigma_x^2}{2}\frac{\sigma_n^2}{w_{opt}^2\sigma_x^2}\sum_{l=-\infty}^{\infty}\rho_{xx}(l)\rho_{nn}(l)$$

with $\rho_{nn}(l) = r_{nn}(l)/\sigma_n^2$.
This last expression is of a product of three terms. The first term is the adaptive constant, the second term is the ratio between the noise and signal power in d and the last term is a measure for similarity of the x time behavior to the n time behavior.

Final misadjustment.
First we will derive an expression for the mean-squared error $E\{e^2[k]\}$.

$$\begin{aligned} E\{e^2[k]\} &= E\{v^2[k]x^2[k]\} - 2E\{v[k]x[k]n[k]\} + E\{n^2[k]\} \\ &= E\{v^2[k]x^2[k]\} - 2E\{\mu\sum_{l=0}^{\infty} p^l x[k-1-l]n[k-1-l]x[k]n[k]\} + \sigma_n^2 \\ &= \frac{\mu}{2}\sum_l r_{xx}[l]r_{nn}[l] - \mu(\sum_l r_{xx}[l]r_{nn}[l] - \sigma_x^2\sigma_n^2) + \sigma_n^2 \\ &= \sigma_n^2(1+\mu\sigma_x^2) - \frac{\mu}{2}\sum_l r_{xx}[l]r_{nn}[l] \\ &= \sigma_n^2\{1+\mu\sigma_x^2(1-\frac{1}{2}\sum_l \rho_{xx}[l]\rho_{nn}[l])\}. \end{aligned}$$

The equation for the relative error is given by:

$$\begin{aligned}\tilde{J}[k] &= \frac{E\{e^2[k]\} - J_{min}}{J_{min}} \\ &= \mu\sigma_x^2(1 - \frac{1}{2}\sum_l \rho_{xx}[l]\rho_{nn}[l])\end{aligned}$$

where

$$J_{min} = E\{e^2[k]\}|_{w[k]=w_{opt}} = E\{n^2[k]\} = \sigma_n^2.$$

So, the final misadjustment $\bar{J}$ is given by:

$$\begin{aligned}\bar{J} &= \lim_{k\to\infty} \tilde{J}[k] \\ &= \mu\sigma_x^2(1 - \frac{1}{2}\sum_l \rho_{xx}[l]\rho_{nn}[l]).\end{aligned}$$

Appendix B

Parameter relation of RLS and LMS algorithm

The purpose of this appendix is to describe the relation of the RLS and LMS algorithm. The weight error is given by:

$$\begin{aligned} v[k] &= w[k] - w_0 \\ &= \frac{P_{xd}[k]}{P_{xx}[k]} - \frac{E\{xd\}}{E\{x^2\}} \\ &= \frac{P_{xd}[k]}{P_{xx}[k]} - \frac{\bar{P}_{xd}}{\bar{P}_{xx}} \\ &= \frac{\bar{P}_{xd} + \tilde{P}_{xd}[k]}{\bar{P}_{xx} + \tilde{P}_{xx}[k]} - \frac{\bar{P}_{xd}}{\bar{P}_{xx}}. \end{aligned}$$

With $\tilde{P}_{\cdot} = P_{\cdot} - \bar{P}_{\cdot}$

$$\begin{aligned} v[k] &\simeq \bar{P}_{xd}(1 + \frac{\tilde{P}_{xd}[k]}{\bar{P}_{xd}})\frac{1}{\bar{P}_{xx}}(1 - \frac{\tilde{P}_{xx}[k]}{\bar{P}_{xx}}) - \frac{\bar{P}_{xd}}{\bar{P}_{xx}} \\ &= \frac{\tilde{P}_{xd}[k]}{\bar{P}_{xx}} - \frac{\tilde{P}_{xx}[k]\bar{P}_{xd}}{\bar{P}_{xx}^2} \\ &= \frac{1}{\bar{P}_{xx}}(\tilde{P}_{xd}[k] - \frac{\tilde{P}_{xx}[k]\bar{P}_{xd}}{\bar{P}_{xx}}) \end{aligned}$$

where

$$\bar{P}_{xx} = E\{x^2\} = \sigma_x^2$$

$$\bar{P}_{xd} = E\{xd\} = w_0\sigma_x^2$$

The adaptive weight is given by:

$$w[k] = \frac{P_{xd}[k]}{P_{xx}[k]}$$

where $P_{xd}[k]$ and $P_{xx}[k]$ are first-order linear time-invariant systems with a nonzero-mean stochastic input described by the difference equations:

$$P_{xd}[k] = \theta P_{xd}[k-1] + (1-\theta)x[k]d[k]$$

$$P_{xx}[k] = \theta P_{xx}[k-1] + (1-\theta)x[k]x[k]$$

Taking the expectation we have

$$\begin{aligned} E\{P_{xd}[k]\} &= (1-\theta)E\{\sum_{l=0}^{\infty}\theta^l x[k-l]d[k-l]\} \\ &= (1-\theta)E\{[x[k]d[k]\}\sum_{l=0}^{\infty}\theta^l \\ &= E\{x[k]d[k]\} = w_0\sigma_x^2. \end{aligned}$$

Using these results, we find that

$$\begin{aligned} v[k] &= \frac{1}{\bar{P}_{xx}}(P_{xd}[k] - w_0\tilde{P}_{xx}[k] - \bar{P}_{xd}) \\ &= \frac{1}{\bar{P}_{xx}}(P_{xd}[k] - w_0\tilde{P}_{xx}[k] - w_0\bar{P}_{xx}) \\ &= \frac{1}{\bar{P}_{xx}}(P_{xd}[k] - w_0 P_{xx}[k]) \\ &= \frac{(1-\theta)}{\sigma_x^2}(\sum_{l=0}^{\infty}\theta^l x[k-l]d[k-l] - w_0\sum_{l=0}^{\infty}\theta^l x^2[k-l]) \\ &= \frac{(1-\theta)}{\sigma_x^2}\sum_{l=0}^{\infty}\theta^l x[k-l]n[k-l] \\ &\simeq \frac{(1-\theta)}{\sigma_x^2}x[k]n[k] + \theta v[k-1]. \end{aligned}$$

So, the adaptive weight equation with the RLS algorithm can be approximate by:

$$v[k] = \theta v[k-1] + \frac{(1-\theta)}{\sigma_x^2}x[k]n[k]. \tag{B.1}$$

In order to get a similar performance to the LMS algorithm we take (compare (A.1) with (B.1))

$$\theta = p = 1 - \mu\sigma_x^2.$$

This shows that, the RLS algorithm, with appropriate choice of its parameters, will have practically the same behavior as the LMS algorithm.

Appendix C

The convergence properties of the RLS algorithm.

In this appendix the convergence properties of the RLS algorithm are derived. Again we consider here the simple model

$$d[k] = w_0 x[k] + n[k],$$

$$v[k] = w[k] - w_0,$$

$$e[k] = d[k] - w[k]x[k] = w_0 x[k] + n[k] - w[k]x[k] = n[k] - v[k]x[k].$$

Weight-error fluctuation.
The update equation for the weight error is:

$$v[k+1] = \theta v[k] + \frac{1-\theta}{\sigma_x^2} x[k]n[k]$$

and

$$E\{v^2[k+1]\} = \theta^2 E\{v^2[k]\} - 2\theta\frac{1-\theta}{\sigma_x^2} E\{v[k]x[k+1]n[k+1]\} + \frac{1-\theta}{\sigma_x^2} E\{x^2[k]n^2[k]\}.$$

With $E\{v^2[k+1]\} \simeq E\{v^2[k]\}$ for a large k this equation can be written as:

$$(1+\theta)E\{v^2[k]\} = \frac{2\theta}{\sigma_x^2} E\{v[k]x[k+1]n[k+1]\} + \frac{1-\theta}{\sigma_x^4} E\{x^2[k]n^2[k]\}.$$

Substituting $v[k]$ results in:

$$\begin{aligned}(1+\theta)E\{v^2[k]\} &= \frac{2\theta(1-\theta)}{\sigma_x^4}\sum_{l=0}^{\infty}\theta^l E\{x[k-l]x[k+1]n[k-l]n[k+1]\} \\ &\quad + \frac{1-\theta}{\sigma_x^4} E\{x^2[k]n^2[k]\} \\ &= \frac{1-\theta}{\sigma_x^4}\sum_{l=-\infty}^{\infty}\theta^{|l|} E\{x[k-l]x[k]n[k-l]n[k]\}.\end{aligned}$$

Let L denote the effective length of $r_{xx}(l)r_{nn}(l)$ and let $1-\theta^L \ll 1$ (the effective length of $p^l \gg$ effective length of $r_{xx}(l)r_{nn}(l)$). Then the weight error fluctuation is given by

$$E\{v^2[k]\} \simeq (\frac{1-\theta}{1+\theta})\frac{1}{\sigma_x^2}\sum_l \rho_{xx}(l)r_{nn}(l) = (\frac{1-\theta}{1+\theta})\frac{\sigma_n^2}{\sigma_x^2}\sum_l \rho_{xx}(l)\rho_{nn}(l).$$

The mean-squared error.
The mean-squared error $E\{e^2[k]\}$ is given by:

$$E\{e^2[k]\} = E\{v^2[k]x^2[k]\} - 2E\{v[k]x[k]n[k]\} + E\{n^2[k]\}$$

Substituting $v[k]$ in the previous equation results in:

$$\begin{aligned}
E\{e^2[k]\} &= E\{v^2[k]x^2[k]\} - 2\frac{1-\theta}{\sigma_x^2}E\{\sum_{l=0}^{\infty}\theta^l x[k-l]n[k-l]x[k]n[k]\} + \sigma_n^2 \\
&= \frac{1-\theta}{(1+\theta)\sigma_x^2}\sum_l r_{xx}[l]r_{nn}[l] - \frac{1-\theta}{\sigma_x^2}(\sum_l r_{xx}[l]r_{nn}[l] + \sigma_x^2\sigma_n^2) + \sigma_n^2 \\
&= \theta\sigma_n^2 + \frac{\theta(\theta-1)}{\sigma_x^2(1+\theta)}\sum_l r_{xx}[l]r_{nn}[l] \\
&= \sigma_n^2(\theta + \frac{\theta(\theta-1)}{1+\theta}\sum_l \rho_{xx}[l]\rho_{nn}[l]).
\end{aligned}$$

So, the mean-squared error is given by:

$$E\{e^2[k]\} = \frac{2\theta^2}{1+\theta}\sigma_n^2\sum_l \rho_{xx}[l]\rho_{nn}[l]$$

Transition behavior.
For $w_0[k] = w_1$ for $k < N$ and $w_0[k] = w_2$ for $k \geq N$ we have

$$\begin{aligned}
E\{w[k]\} &= \frac{(1-\theta)}{\sigma_x^2}\sum_{l=0}^{\infty}\theta^l E\{x[k](w_0[k]x[k] + n[k])\} \\
&= (1-\theta)\sum_{l=0}^{N-1}\theta^l w_1 + \sum_{l=N}^{k}\theta^l w_2 \\
&= w_2 + (w_1 - w_2)\theta^k.
\end{aligned}$$

Again, this shows that taking $\theta = 1 - \mu\sigma_x^2$ gives the similar transient behavior for LMS and RLS algorithm.

Appendix D

C programs "LMS algorithm"

The following program is a C-implementation of the LMS program as discussed in Section 7.1. This program essentially consists of two instructions only: the calculation of the error and the update of the weight.

The libraries PDADA41, PDINT40, PDTMR and PDTYPE are from VIRTUOSO-libraries and are needed for the A/D and D/A interface [10].

```
#include "iface.h"
#include "node1.h"
#include "_stdio.h"
#include "string.h"
#include "allnodes.h"

#include "PDADA4A1.H"
#include "PDINT40.H"
#include "PDTMR40.H"
#include "PDTYP40.H"

#define CPU_CLOCK       60E6     /* 50 MHz */
#define FREQ            1000     /* 1khz       */
#define Mu              1E-3     /*  The step size of LMS algorithm  */

volatile        int     Flag;
volatile        float   in_r, in_b;

float           Err, Alpha=0.803;

ADA4A1Struct    ADA1;           /* define ADA card structure  */

void c_int02 ( void )
 {
  in_r=ADA4A1_READ_FLOAT ( ADA1.Ch1 );  /* read float from channel 1 */
  in_b=ADA4A1_READ_FLOAT ( ADA1.Ch2 );  /* read float from channel 2 */

  ADA1.Cnv->Data = 0xFF;                /* start conversion        */
```

```
  Flag=1;


void prog1 ( void )
{
  if(ADA4A1Init(&ADA1, 0xF0000000)==OK);  /* test ADA1 card */
    {
     ADA4A1Calibrate(&ADA1);              /* calibrate ADA1 */
    }

  if (KS_EnableISR(0, c_int02) != RC_OK)
   {
     printf("\n Unable to install interrupt routine!");
   }

  Tmr0 ->Period=CPU_CLOCK/(4*FREQ);
  Tmr0 ->Control.Data = TMR_START;

while(1)
 {
     while(FLAG!=1);
      {
Err=in_r - Alpha*in_b;
Alpha=Alpha+Mu*Err*in_b;

ADA4A_WRITE_FLOAT(ADA1.Ch3, Err);  /* write float to channel 3  */

FLAG=0;
      }
 }
  printf("\n\n");

  printf("\n End of LMS algorithm");
}
```

The following program was written to file the signals from the sensor on the harddisk as used in test 1 (Chapter 8). Two ADA card structure (ADA1 and ADA2) are define in order to have four analog inputs (for signal r, b and y and one reserved). These two cards have different base addresses (see [10]) which are defined at the initialization (ADA4A1Init()).

```
#include "iface.h"
#include "node1.h"
#include "_stdio.h"
#include "string.h"
```

```
#include "allnodes.h"

#include "PDADA4A1.H"
#include "PDINT40.H"
#include "PDTMR40.H"
#include "PDTYP40.H"

#define CPU_CLOCK       60E6        /* 50 MHz */
#define FREQ            1000        /* 1khz       */
#define Mu              1E-3        /* step size */
#define n1              1000        /* number of samples to save */

volatile        int     Flag;
volatile        float   in_r, in_b, in_y, in_e;

float           Err, Alpha=0.803;
float           out_r[n1], out_b[n1], out_y[n1], out_e[n1];

static  FILE    *fp1, *fp2, *fp3, *fp4;

ADA4A1Struct    ADA1;                       /* define ADA1 card structure  */
ADA4A1Struct    ADA2;                       /* define ADA2 card structure  */


fp1=fopen("out_r.met","r");
fp2=fopen("out_b.met","r");
fp3=fopen("out_y.met","r");
fp4=fopen("out_e.met","r");

void c_int02 ( void )
 {
  in_r=ADA4A1_READ_FLOAT ( ADA1.Ch1 );    /* read float from channel 1 */
  in_b=ADA4A1_READ_FLOAT ( ADA1.Ch2 );    /* and channel 2 of ADA1 card*/

  in_Y=ADA4A1_READ_FLOAT ( ADA2.Ch1 );    /* read float from channel 1 */
  /* of ADA2 card                    */

  ADA1.Cnv->Data = 0xFF;                  /* start conversion (ADA1) */
  ADA2.Cnv->Data = 0xFF;                  /* start conversion (ADA2) */

  Flag=1;                               /* set the flag in ready state */

  }

void prog1 ( void )
{
  if(ADA4A1Init(&ADA1, 0xF0000000)==OK);  /* test ADA1 card */
```

```
    {
     ADA4A1Calibrate(&ADA1);              /* calibrate ADA1 */
    }
  if(ADA4A1Init(&ADA2, 0xF0000000)==OK);  /* test ADA2 card */
    {
     ADA4A1Calibrate(&ADA2);              /* calibrate ADA2 */
    }

  if (KS_EnableISR(0, c_int02) != RC_OK)
   {
     printf("\n Unable to install interrupt routine!");
   }

  Tmr0 ->Period=CPU_CLOCK/(4*FREQ);      /* set timer0 period time */
  Tmr0 ->Control.Data = TMR_START;       /* start timer0           */

while(i=n1)
 {
     while(FLAG!=1);
      {
Err=in_r - Alpha*in_b;          /* calculate the error signal */
Alpha=Alpha+Mu*Err*in_b;        /* and the adaptive weight   */

out_r[i]=in_r;                   /* put the signals r, b, y   */
out_b[i]=in_b;                     /* and Err in vectors      */
out_y[i]=in_y;
out_e[i]=Err;

FLAG=0;                             /* set the flag in wait state*/
      }
 }
  for(i=0;i<n1;i++)
  {
   fprintf(fp1,"%f\n".out_r[i]);     /* write the vectors on file */
   fprintf(fp2,"%f\n".out_b[i]);
   fprintf(fp3,"%f\n".out_y[i]);
   fprintf(fp4,"%f\n".out_e[i]);
  }
  printf("\n\n");

  printf("\n End of LMS algorithm");
}
```

Appendix E

Assembler program

The assembler code of the first C program in Appendix D is given below.

```
*******************************************************
* FUNCTION DEF : _c_int02
*******************************************************
_c_int02:
PUSH    ST
PUSH    FP
LDI     SP,FP
PUSH    R0
PUSHF   R0
PUSH    R1
PUSHF   R1
PUSH    AR0
PUSH    AR1
PUSH    AR2
PUSH    R9
PUSHF   R9
*
* R9    assigned to temp var  C$1
*
*** 27 ---------- in_r = -((float)((*ADA1.Ch1.Data&0xfff0)<<16)*C$1);
.sym    C$1,23,6,4,32
.line   4
LDA     @_ADA1+1,AR0
LDI     @CONST+0,R0
AND     R0,*AR0,R1
LSH     16,R1,R1
FLOAT   R1
NEGF    R1
LDF     @CONST+1,R9
MPYF    R9,R1
STF     R1,@_in_r
*** 28 ---------- in_b = -((float)((*ADA1.Ch2.Data&0xfff0)<<16)*C$1);
.line   5
LDA     @_ADA1+4,AR1
AND     R0,*AR1,R1
LSH     16,R1,R1
```

```
FLOAT   R1
NEGF    R1
MPYF    R9,R1
STF     R1,@_in_b
*** 30 ----------  ADA1.Cnv->Data = 255u;
.line   7
LDA     @_ADA1+13,AR2
LDI     255,R1
STI     R1,*AR2
*** 31 ----------  Flag = 1;
.line   8
STIK    1,@_Flag
***    ----------  return;
EPI0_1:
.line   9
POPF    R9
POP     R9
POP     AR2
POP     AR1
POP     AR0
POPF    R1
POP     R1
POPF    R0
POP     R0
POP     FP
POP     ST
RETI

.endfunc        32,000800703H,0

.sym    _prog1,_prog1,32,2,0
.globl  _prog1

.func   34
*******************************************************
* FUNCTION DEF : _prog1
*******************************************************
_prog1:
PUSH    FP
LDI     SP,FP
PUSH    R4
*
* R2    assigned to temp var  K$15
* R3    assigned to temp var  K$14
* R4    assigned to temp var  C$2
* AR2   assigned to temp var  C$1
* AR2   assigned to temp var  U$22
* R9    assigned to temp var  Y$0
* R10   assigned to temp var  U$12
*
*** 37 ----------  C$2 = &ADA1;
.sym    U$22,10,20,4,32
.sym    Y$0,23,6,4,32
```

```
.sym    U$12,24,6,4,32
.sym    K$15,2,6,4,32
.sym    K$14,3,6,4,32
.sym    C$1,10,24,4,32,$_Unknown_Tag_Name_$
.sym    C$2,4,24,4,32,$_Unknown_Tag_Name_$
.line   4
LDI     @CONST+2,R4
*** 37 ----------  ADA4A1Init(C$2, 0xf0000000u);
LDA     R4,AR2
LDI     @CONST+3,R2
CALL    _ADA4A1Init
*** 37 ----------  ADA4A1Calibrate(C$2);
LDA     R4,AR2
CALL    _ADA4A1Calibrate
*** 44 ----------  flag = 0;
.line   11
STIK    0,@_flag
*** 47 ----------  if ( !KS_EnableISR(0, &c_int02) ) goto g2;
.line   14
LDA     0,AR2
LDI     @CONST+4,R2
CALL    _KS_EnableISR
CMPI    0,R0
BZ      L2
*** 49 ----------  printf("\nUnable to install interrupt routine !");
.line   16
LDI     @CONST+5,R0
PUSH    R0
CALL    _printf
SUBI    1,SP
L2:
***     ----------g2:
*** 53 ----------  C$1 = Tmr0;
.line   20
LDA     @_Tmr0,AR2
*** 53 ----------  C$1->Period = 7500u;
LDI     7500,R0
STI     R0,*+AR2(8)
*** 54 ----------  C$1->Control.Data = 705u;
.line   21
LDI     705,R1
STI     R1,*AR2
***     ----------  U$22 = ADA1.Ch3.Data;
LDA     @_ADA1+7,AR2
***     ----------  U$12 = Alpha;
LDF     @_Alpha,R10
***     ----------  K$14 = 0.001F;
LDF     @CONST+6,R3
***     ----------  K$15 = 2.147483647648156e8F;
LDF     @CONST+7,R2
L4:
***     ----------g4:
*** 61 ----------  if ( flag != 1 ) goto g4;
```

```
.line   28
LDI     @_flag,R0
CMPI    1,R0
BNZ     L4
*** 63 ----------  Err = Y$0 = in_r-U$12*in_b;
.line   30
MPYF    @_in_b,R10
SUBRF   @_in_r,R10
LDF     R10,R9
STF     R10,@_Err
*** 64 ----------  U$12 = Alpha += in_b*K$14*Y$0;
.line   31
LDF     @_in_b,R0
MPYF    R3,R0,R10
MPYF    R9,R10
ADDF    @_Alpha,R10
STF     R10,@_Alpha
*** 66 ----------  *U$22 = (int)(Y$0*K$15)>>16;
.line   33
MPYF    R2,R9,R0
FIX     R0,R1
NEGF    R0
FIX     R0
NEGI    R0
LDILE   R0,R1
ASH     -16,R1,R0
STI     R0,*AR2
*** 68 ----------  goto g4;
.line   35
B       L4
EPI0_2:
.line   38
LDI     *-FP(1),R1
LDI     *FP,FP
POP     R4
SUBI    2,SP
B       R1
.endfunc        71,000000010H,0

.sym    _ADA1,_ADA1,8,2,576,.fake47
.globl  _ADA1
.bss    _ADA1,18

.sym    _in_b,_in_b,6,2,32
.globl  _in_b
.bss    _in_b,1

.sym    _in_r,_in_r,6,2,32
.globl  _in_r
.bss    _in_r,1

.sym    _flag,_flag,4,2,32
.globl  _flag
```

```
.bss    _flag,1

.sym    _Err,_Err,6,2,32
.globl  _Err
.bss    _Err,1
*******************************************************
* DEFINE STRINGS                                      *
*******************************************************
.sect   ".const"
SL0:    .byte   10,"Unable to install interrupt routine !",0
*******************************************************
* DEFINE CONSTANTS                                    *
*******************************************************
.bss    CONST,8
.sect   ".cinit"
.word   8,CONST
.word   65520           ;0
.float  4.656612873840332e-9;1
.word   _ADA1           ;2
.word   -268435456      ;3
.word   _c_int02        ;4
.word   SL0             ;5
.float  1.0e-3          ;6
.float  2.147483647648156e8;7
*******************************************************
* UNDEFINED REFERENCES                                *
*******************************************************
.globl  _KS_EnableISR
.end
```

Bibliography

[1] Haykin, S.
Adaptive filter theory.
Englewood Cliffs, NJ: Prentice-Hall, 1991.

[2] Butterweck, H.J.
Iterative analysis of the steady-state weight fluctuations in LMS-type adaptive filters.
EUT Report 96-E-299. Eindhoven: Eindhoven University of Technology, 1996.

[3] Charles, W.T.,
Discrete random signals and statistical signal processing.
Englewood Cliffs, NJ: Prentice-Hall, 1992.

[4] Sommen, P.,
Tijddiscrete signaalbewerking (TDS) (5L090).
Eindhoven: Eindhoven University of Technology, 1993. Syllabus.

[5] Ritzerfeld, J., H. Hegt, P. Sommen and H. van Meer,
Realisering van digital signaalbewerkende systemen (5N290).
Eindhoven: Eindhoven University of Technology, 1993. Syllabus.

[6] Buël, C.A.H. van,
Introduction to the signal processing workplace.
Eindhoven: Eindhoven University of Technology, 1996. MSc. Thesis.

[7] Texas Instruments, October 1991.
TMS320 Floating-point DSP Optimizing C Compiler. User's Guide

[8] Texas Instruments, May 1991.
TMS320C4x User's Guide.

[9] Eonic Systems.
Virtuoso User Manual.
Eonic Systems Inc, 1995.

[10] Prodrive.
TIMEX Modular DSP System User Manual.
Prodrive, February 1996.

[11] Eonic Systems.
Virtuoso Modulo 1.
Eonic Systems Inc, 1995.

List of symbols

b	the blue signal.
b'_0	the noise-free blue signal.
$C(.)$	the coherence function.
D	the sampling time.
e_1	the error signal with b in regression.
e_2	the error signal with r in regression.
$E\{x\}$	the expectation for $x[k]$.
$E\{x^2\}$	the variance for $x[k]$.
$E\{xy\}$	the covariance for $x[k]$.
n_1, s_1	noise in the red range.
n_2, s_2	noise in the blue range.
r	the red signal.
r'_0	the noise free red signal.
$r_{xx}[l]$	the autocorrelation function of x.
$S_{xx}(\omega)$	the power spectral density function for $x[k]$.
$S_{xx}(\omega)$	the cross-power spectral density function for $x[k]$ and $y[k]$.
T_{xy}	the transfer function (S_{xy}/S_{xx}).
v	the weight error.
w	the adaptive weight.
w_{opt}	the Wiener solution.
α	the adaptive weight with b in regression.
α_{opt}	the optimal solution with b in regression.
α_0, β_0	the color coefficient $(\alpha_0 = 1/\beta_0)$.
β	the adaptive weight with r in regression.
β_{opt}	the optimal solution with r in regression.
θ	parameter (in estimation).
μ	the step size in LMS algorithm.
$\rho_{xx}[l]$	the normalized autocorrelation function of x.
σ_x	the variance of $x[k]$.
σ_n	the variance of $n[k]$.
τ	the integration constant.

List of Figures