Cancellation for periodic disturbances

Cancellation of periodic disturbances

J.H. Bonsel

DCT-2002-56

Internship at Océ-Technologies B.V.
Department Research & Development

Coach: Dr. ir. B.B. Goeree (Océ-Technologies B.V.)
Supervisor: Prof. dr. H. Nijmeijer (TU Eindhoven)

Venlo, July 2002

Summary

In a printer, an image is initially brought onto a belt before it is transferred to a sheet of paper. Although the position of this belt is controlled, position errors occur. These errors are partly of repetitive nature. Due to these errors, an image is printed too low or too high on a sheet of paper. The goal of the research is to design an algorithm that predicts the position error of this belt, so that the image can be brought onto the belt at an adjusted time, in order to compensate the position error.

The position of the belt is measured by an encoder and is compared with a reference position, resulting in an error. Some experimentally obtained error data series are analysed. Powerspectra of the error data show that a significant part of the error power is present at only a few fixed frequencies. Furthermore, these repetitive parts of the error signal have a shape similar to a sine wave.

Two control strategies that are able to counteract periodic disturbances are repetitive control and active noise control. Repetitive control uses a memory loop to generate periodic signals that can be used for compensation. In order to predict the position error of the belt, a repetitive control system is designed and extended with some filters. The designed system is tested on the experimentally obtained error data. The results show that a reduction in error power can be obtained. Active noise control is a signal processing strategy in which a digital filter processes a reference signal, in order to compensate the error signal. An adaptive algorithm adapts the filter coefficients at runtime. An optimised active noise control system is tested on the experimentally obtained error data. The results show that a larger error power reduction is obtained than with the repetitive control configuration.

In order to test whether a significant improvement in registration accuracy is made, the designed system is tested on an experimental set-up. The choice is made to use the active noise control system, because this strategy results in a larger error power reduction, requires less computational effort and is easier to implement than the designed repetitive control system. The designed algorithm is translated from Matlab to the C language and is integrated with the software that runs on the experimental set-up. Experiments show that the online performance of the algorithm is similar to the offline performance, which means that the algorithm is able to predict a part of the position error of the belt. However, using the algorithm to adjust the moment of start of page does not influence the registration accuracy significantly. The explanation for this is probably that the experimental set-up produces too many other disturbances that affect the registration accuracy.

List of symbols

Symbol	Meaning	Unit
ω	angular frequency	rad/s
μ	convergence rate	-
ξ	mean square error	mm^2
$\hat{\xi}$	mean square error estimate	mm^2
ω_0	frequency of primary noise signal	rad/sample
ϕ_e	phase of primary noise signal	rad
A	amplitude of reference signal	-
A_e	amplitude of primary noise signal	-
b	dynamic compensator	-
C	controller	-
d	disturbance signal	-
$E[\,]$	expectation operator	-
e	servo error / measured error	- / mm
e_c	compensated error	mm
H	transfer function	-
k	gain	-
L	number of filter coefficients	-
M	number of frequencies	-
N	number of samples	-
n	sample number	-
$\underline{p}$	cross-correlation vector	mm^2
P	process	-
P_r	signal power	mm^2
q	low-pass filter	-
R	auto-correlation matrix	mm^2
$\underline{r}$	reference signal vector	-
r	reference signal	-
r_l^m	reference signal	-
T	sampling time	s
$\underline{w}$	filter coefficient vector	-
w_l	filter coefficient	-
w_l^o	optimal filter coefficient	-
$\underline{w}^o$	optimal filter coefficient vector	-
y	output signal	-
y^m	individual output	-
z	variable in the z-transform	-

Contents

Summary ii

List of symbols iii

1. Introduction 1

2. Error data analysis 2

3. Repetitive control 4
3.1 Memory loop filtering 4
3.2 Repetitive control: general case 5
3.3 Application of repetitive control 7

4. Active noise control 13
4.1 Narrowband feedforward ANC 13
4.2 Adaptive algorithm 14
4.3 Single frequency ANC 16
4.4 Multiple frequency ANC 17
4.5 Application of ANC 18

5. Comparison of repetitive control with active noise control 22

6. Implementation 24
6.1 Composition of the functions 24
6.2 Experimental results 24

Conclusions and recommendations 27

Appendix A: Histogram error signal 28

Appendix B: Matlab functions 29

Appendix C: C functions 31

References 37

1. Introduction

The demands made on the accuracy with which an image is printed on a sheet of paper get higher and higher. One of the errors that occur is that an image is printed too high or too low on a sheet of paper. This error is partly of repetitive nature. If this error can be predicted, the start of page signal can be adjusted, so that the image is printed a little earlier or later to compensate the error.

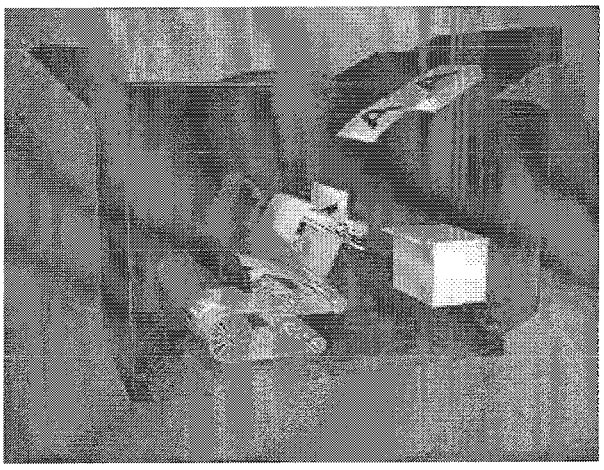

Figure 1.1: Inside of an océ printer

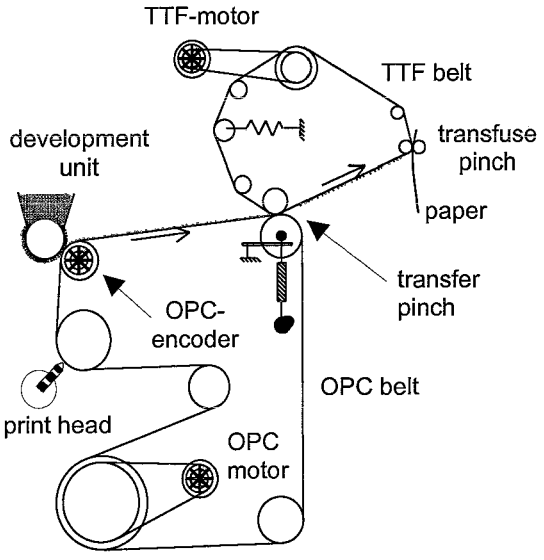

Figure 1.2: Schematic representation

In Figure 1.1 a part of the inside of an océ printer is depicted. Two belts are visible. The lower belt is the organic photo conductor belt (OPC). The upper is the transfer-and-transfuse belt (TTF). Initially a charge is brought onto the OPC belt. After that, the charge is exposed in order to create the image in the charge. Next, the charge will attract the toner and a visible image is developed. Then the image is transferred to the TTF belt and the OPC belt is discharged. Finally, the TTF belt transfers the toner image to a sheet of paper.

Figure 1.2 is a schematic representation of the OPC and TTF belt. The OPC belt is driven by the OPC motor and its position is measured by an encoder every 2 ms. The OPC belt is assumed to be rigid. Although the belt is controlled, position errors occur, which are partly repetitive. Possible causes of these errors can be eccentric rollers or collision of the paper with the transfuse pinch. Earlier performed experiments show that these position errors result in the image being printed either too low or too high on the sheet of paper. If one can predict the position error of the OPC belt, the print head can bring the image onto the OPC belt a little earlier or later in order to compensate this error.

The goal of the research is to design an algorithm that predicts the position error of the OPC belt as accurate as possible, so that it can be compensated. In order to do this, first some error data obtained from an experimental set-up will be analysed. The main purpose of this analysis is to determine what part of the error signal is repetitive and what part is random noise. After that, two control strategies that are able to counteract periodic disturbances will be treated: repetitive control and active noise control. The principles of these strategies will be discussed and the strategies will be adapted to the problem. They will be tested and optimised by means of simulations performed on error data. Both strategies will be compared in order to find the most suitable solution to this problem. Finally the most suitable strategy will be implemented on the experimental set-up. Measurements will be made in order to determine whether an improvement is made.

2. Error data analysis

Before the design of an algorithm that predicts an error can be started, some insight into the error signal must be obtained. Especially the frequencies the error signal consists of are of great importance. In order to make error prediction possible, a significant part of the error power must be present at only a few frequencies. In this chapter some experimentally obtained error data series will be analysed.

In the experimental set-up an encoder measures the belt position with a sampling frequency of 500 Hz. The measured position can be compared with a reference position, resulting in an error. Ten data series are available, each of one minute length. All data series are obtained from the same experimental set-up on different days. A straightforward way to analyse the error data is to construct powerspectra. Figure 2.1 shows the powerspectra of two arbitrary data series. Because nearly all power is present in frequencies up to 15 Hz, the powerspectra are plotted up to this frequency. It is visible that the powerspectra show peaks at some frequencies. The two powerspectra are similar in a way that peaks are visible at the same frequencies. However, the heights of the peaks in the two powerspectra differ. The same similarity holds for the remaining data series.

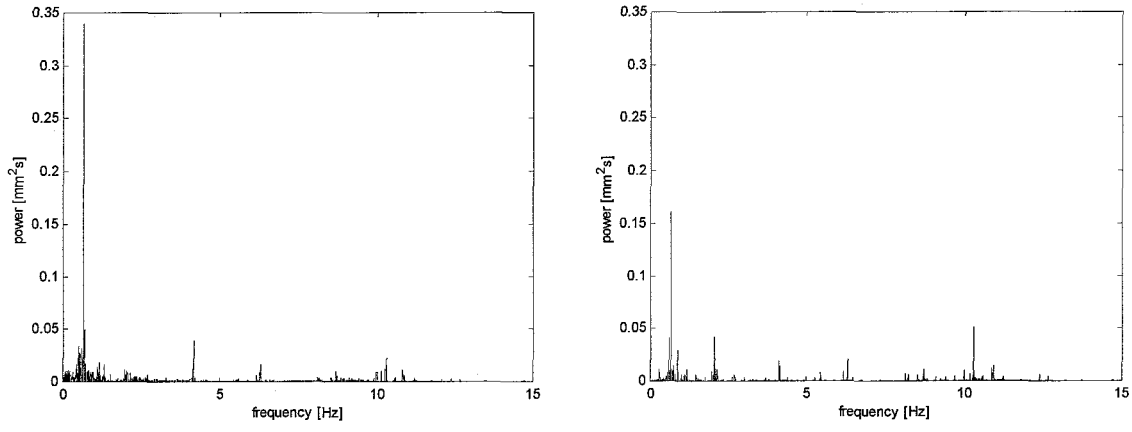

Figure 2.1: Powerspectra of experimentally obtained error data

The powerspectra show large peaks at 0.66 Hz. A question that arises is how much the error power will decrease if this frequency is filtered out. In order to answer it, first the total error power is determined. Second, the signal is averaged over the period corresponding to 0.66 Hz. This is done by dividing the signal in parts of 1/0.66 = 1.51 seconds, as illustrated in figure 2.2, adding the individual parts, as illustrated in figure 2.3, and dividing the resulting signal by the number of parts.

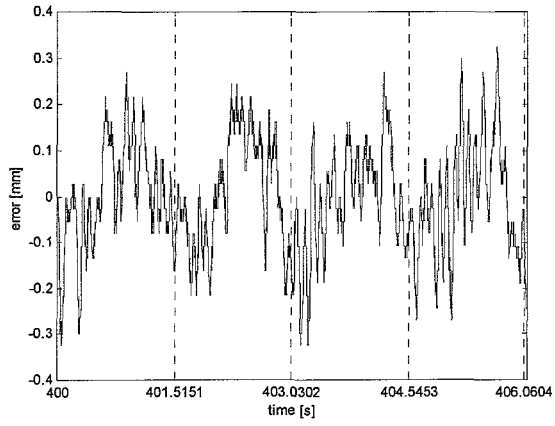

Figure 2.2: Divide signal in parts

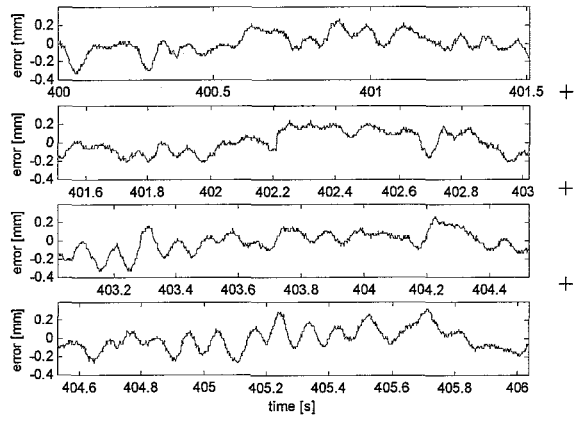

Figure 2.3: Add individual parts

Figure 2.4 shows signals that are obtained by averaging the error data of one series over the period corresponding to 0.66 Hz and 4.15 Hz respectively. One can see that the part of the error signal with frequency 0.66 Hz is approximately a sine wave and the part with frequency 4.15 Hz is almost a perfect sine wave, so the problem might very well lay with eccentric rollers.

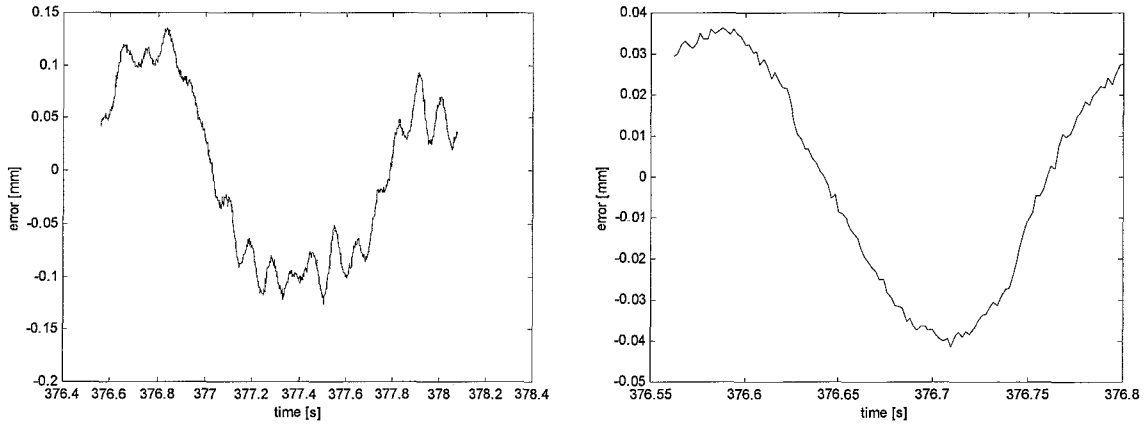

Figure 2.4: Averaged signals

Finally the averaged signal is subtracted from the original signal and the signal power is determined again. Subtracting the over 0.66 Hz averaged signal from the original signal and recalculating the signal power results on average in a decrease of approximately 25%. If this procedure is repeated for the five most powerful frequencies, an error power reduction of approximately 50% is obtained.

3. Repetitive control

A control strategy that is able to counteract periodic disturbances is repetitive control. It is based on the internal model principle, which states that a model of the disturbance should be included in the feedback loop in order to achieve asymptotic tracking and disturbance rejection. The main part of a repetitive controller is the memory loop, which will be treated first. Second, the general repetitive control configuration will be looked at. Third, the repetitive control configuration will be adapted to the problem. Repetitive control will be treated in discrete-time because the system will use the discrete signal from the encoder.

3.1 Memory loop filtering

The main part of a repetitive controller is the memory loop, which can be seen as a signal generator that will generate arbitrary periodic signals. A memory loop is a digital filter that consists of a series of time delays with a positive feedback loop. Figure 3.1 shows the configuration of a memory loop. It can generate discrete-time signals with fundamental period NT in which T is the sampling time of the digital filter and N is an integer representing the number of samples in the memory loop.

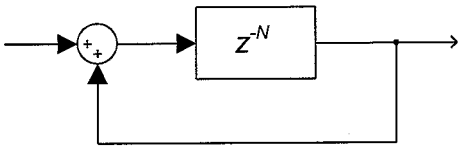

Figure 3.1: Memory loop

The transfer function of this configuration can be expressed as:

$$H = \frac{z^{-N}}{1 - z^{-N}} = \frac{1}{z^N - 1} \tag{3.1}$$

From the transfer function can be seen that the memory loop has N poles uniformly distributed on the unit circle, which implies that the system is on the boundary of stability. This is shown graphically in figure 3.2. The figure shows a pole-zero map of a memory loop with N = 10 samples delay and a sampling time of T = 0.01 s. The magnitude plot of this memory loop is shown in figure 3.3. At the fundamental frequency, $1/NT$ = 1/(10*0.01) = 10Hz, and at its multiples, 'infinite' peaks are visible. The variation in height of the peaks is the effect of inaccuracies of the solver.

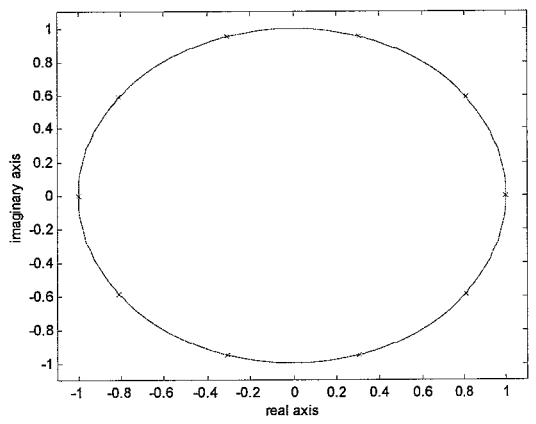

Figure 3.2: Pole-zero map of memory loop

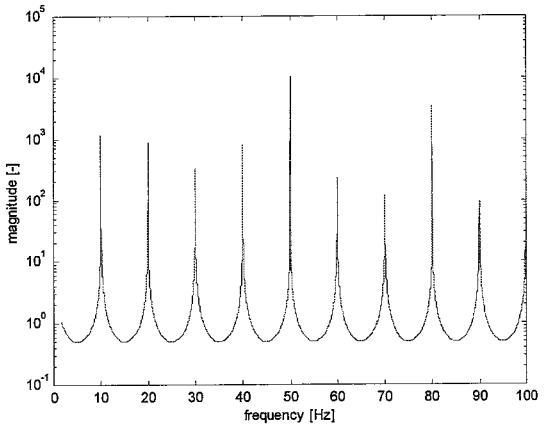

Figure 3.3: Magnitude plot of memory loop

3.2 Repetitive control: general case

The most commonly used repetitive control configuration is shown in figure 3.4. Here the memory loop is connected to the control loop in an add-on manner. This configuration is used to either suppress periodic disturbances *d* or to track periodic reference signals *r*. The blocks *P* and *C* represent the process to be controlled and the conventional controller respectively.

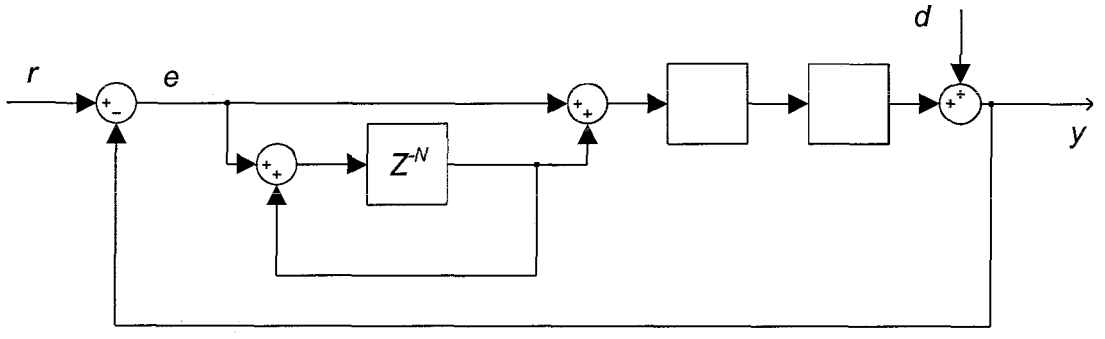

Figure 3.4: Basic repetitive control configuration

The servo error *e* of the system can be expressed as:

$$e(z^{-1}) = \left\{\frac{1}{1+CP}\right\}\frac{1-z^{-N}}{1-z^{-N}\left(1-\frac{CP}{1+CP}\right)}(r-d) \tag{3.2}$$

According to the small gain theorem the system is asymptotically stable if [1]:

$$\frac{1}{1+CP} \tag{3.3}$$

is asymptotically stable and if

$$\left|1-\frac{CP}{1+CP}\right| < 1 \tag{3.4}$$

In general this last condition is not satisfied. In order to guarantee stability a dynamic compensator *b* is included in the system, as shown in figure 3.5.

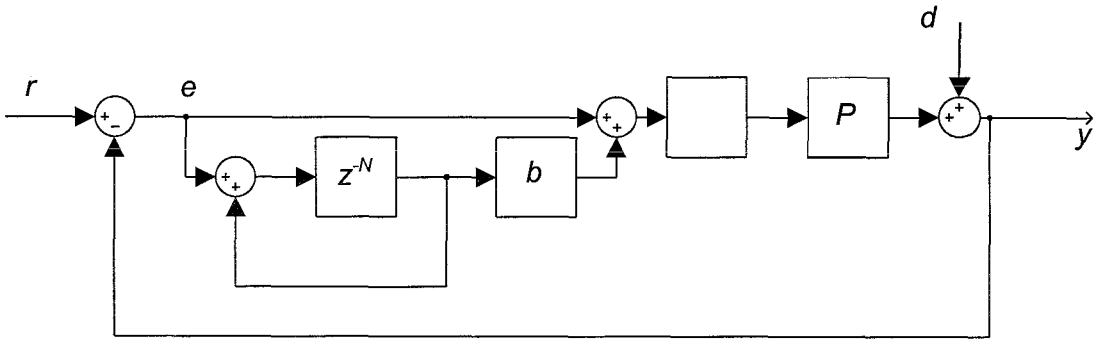

Figure 3.5: Repetitive control configuration with dynamic compensator

The servo error (3.2) now becomes:

$$e(z^{-1}) = \left\{\frac{1}{1+CP}\right\} \frac{1-z^{-N}}{1-z^{-N}\left(1-b\dfrac{CP}{1+CP}\right)}(r-d) \tag{3.5}$$

Stability condition (3.4) changes into:

$$\left|1-b\frac{CP}{1+CP}\right| < 1 \tag{3.6}$$

A straightforward way to obtain stability is to take b as follows:

$$b = k\frac{1+CP}{CP} \tag{3.7}$$

with $0<k<2$. However, if the transfer function $\dfrac{CP}{1+CP}$ contains unstable zeros, these zeros will appear as unstable poles in b. Besides that, the differentiating nature of an inverse system might amplify high-frequency disturbances [1]. In this case, usually a zero-phase cancellation filter is used [2].

If a suitable dynamic compensator has been found, stability problems may still arise due to unmodelled high-frequency dynamics. In order to overcome these problems a low-pass filter q is added. This filter will weaken the effect of the memory loop at higher frequencies. The side effect of this extension is that tracking will be lost in the high frequency range beyond the passband. The cut-off frequency should thus always be chosen a little higher than the signal bandwidth of the periodic signal affecting the system. Figure 3.6 shows the repetitive control configuration including the low-pass filter q.

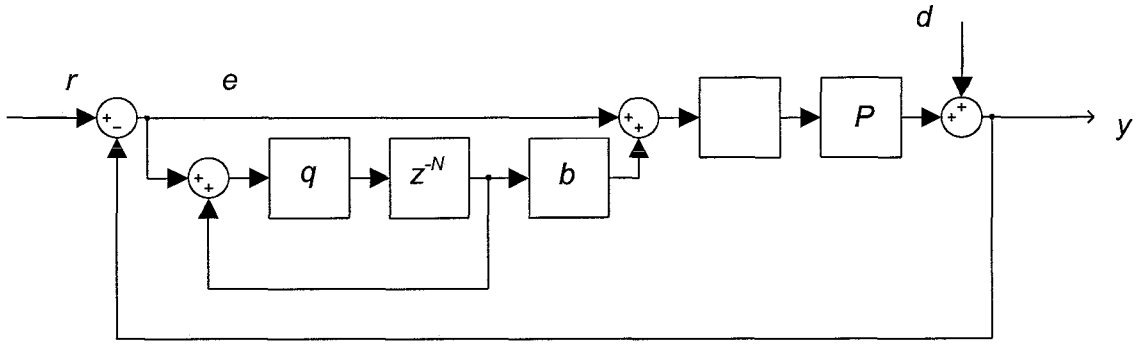

Figure 3.6: Repetitive control configuration with dynamic compensator and low-pass filter

The servo error (3.2) changes into:

$$e(z^{-1}) = \left\{\frac{1}{1+CP}\right\} \frac{1-z^{-N}}{1-z^{-N}q\left(1-b\dfrac{CP}{1+CP}\right)}(r-d) \tag{3.8}$$

Stability condition (3.4) becomes:

$$\left| q\left(1 - b\frac{CP}{1+CP}\right)\right| < 1 \qquad (3.9)$$

3.3 Application of repetitive control

In order to use repetitive control for the specific problem of error prediction, the general configuration must be adapted. Because the system only has to predict an error value, the adapted configuration does not contain a controller *C* and a process *P* and can be simplified to figure 3.7.

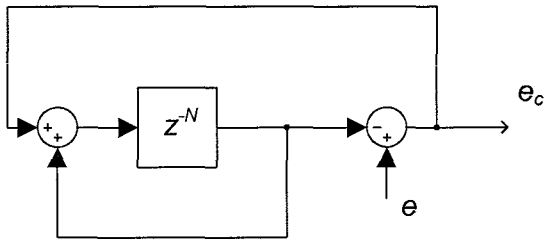

Figure 3.7: Repetitive control configuration for error prediction

The input of the system is the measured error *e*. The memory loop generates a signal that compensates for a specific repetitive part of the error signal. The output of the system is the compensated error signal e_c, which should become zero in the optimal case. In fact the system predicts (a part of) the error signal and compensates for it.

The transfer function of this configuration can be written as:

$$H = \frac{z^N - 1}{z^N} \qquad (3.10)$$

All poles of the system lie at the origin, so the system is stable. The system's performance can be analysed by considering its magnitude plot, which is shown in figure 3.7a. Figure 3.7b shows the same plot, zoomed in around magnitude equal to zero. The plots are made with $N = 50$ samples delay and a sampling time T of 0.002 s.

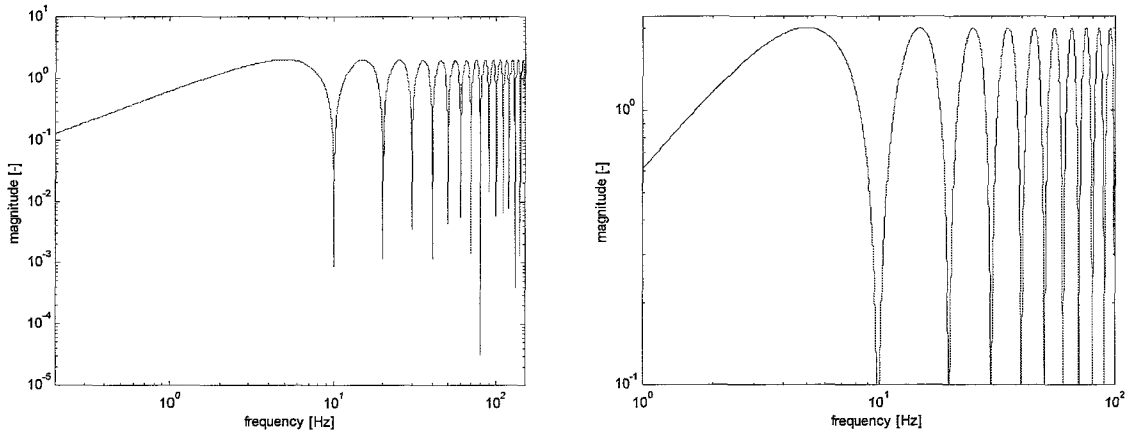

Figure 3.7a/b: Magnitude plots repetitive control system for error prediction

The figures show that the magnitudes of the fundamental frequency ($1/NT$ = 10 Hz) and its multiples are very small, so these frequencies are suppressed. Furthermore, frequencies between multiples of the fundamental frequency have a magnitude greater than 1, so these frequencies will be amplified by repetitive control. This might be a drawback if the error signal contains frequencies that differ from the fundamental frequency or multiples thereof.

The system is tested with a sine wave as input signal. First a frequency of 10 Hz is taken. This is the frequency for which the system is designed. Figure 3.8a shows the results of the test. As expected, after one period of the sine wave, the system has totally converged and the compensated error e_c is zero. Next a slightly different frequency of 11 Hz is used (figure 3.8b). The compensated error does not converge to zero anymore, but is still smaller than the uncompensated error. Finally a frequency of 15 Hz is used as an input signal (figure 3.8c). Now the compensated error is even bigger than the uncompensated error. These results agree with the conclusions drawn from the magnitude plots.

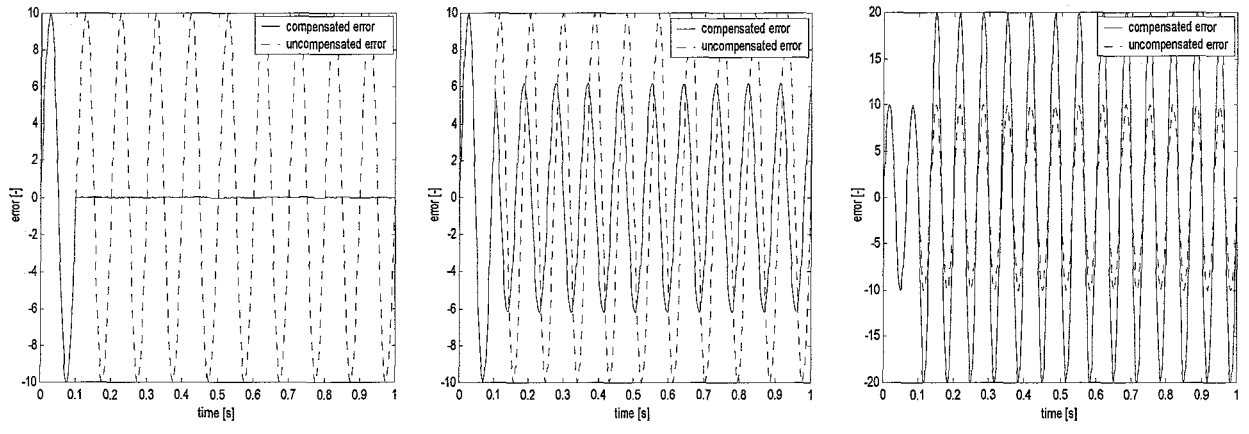

Figure 3.8a/b/c: Error as a function of time for sine inputs of respectively 10 Hz, 11 Hz and 15 Hz

Now the system is ready to be tested on the real data obtained from the experimental set-up. Data analysis performed in chapter 2 showed that 0.66 Hz is the most powerful frequency. Therefore the delay in the memory loop is chosen 1/0.66 Hz = 1.515 s. The sampling time T of the measurements amounts to 0.002 s, so the length of the memory loop N is 1.515/0.002 = 758 samples. Simulations are performed on all available data-series. The resulting compensated error is compared with the uncompensated error, with poor results. The power of the compensated power is on average even 50% higher than the power of the uncompensated error. However, the peak in the powerspectrum at 0.66 Hz has seriously decreased, as can be seen from figure 3.9. This figure shows the powerspectrum of one arbitrary data series together with its compensated powerspectrum.

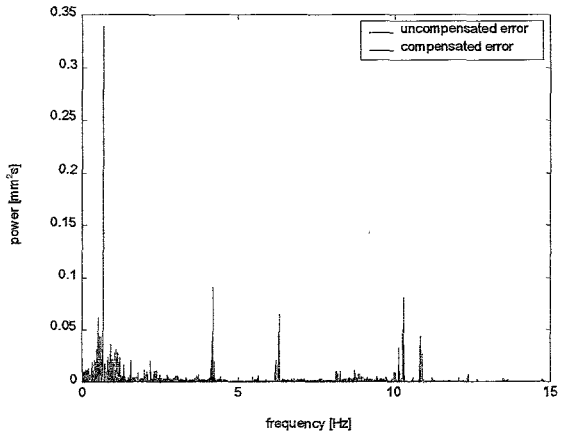

Figure 3.9: Powerspectrum of compensated and uncompensated error signal

A simple modification of the system is carried out in order to improve the results. A gain k is added in the memory loop as shown in figure 3.10.

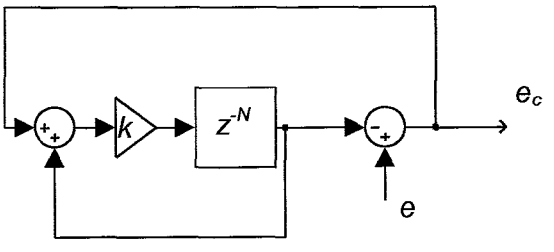

Figure 3.10: Repetitive control configuration for error prediction with additional gain

Transfer function (3.10) changes into:

$$H = \frac{z^N - k}{z^N} \tag{3.11}$$

By adjusting the gain k, the effect of the memory loop can be strengthened or weakened. The latter results in a magnitude plot with less high maxima and less low minima. The system can be optimised by adjusting the gain until the power of the compensated error is lowest. The optimal gain is found to be approximately 0.25. Figure 3.11 shows the magnitude plot of the transfer function with $k = 0.25$ together with the one with $k = 1$. It is visible that the system with $k = 0.25$ suppresses the fundamental frequency and its multiples less than the system with $k = 1$ does. However, frequencies between the fundamental frequency and its multiples are less amplified. Simulations performed on the same data series result in a compensated error power that is approximately equal to the uncompensated error power. Figure 3.12 shows the powerspectra of the uncompensated error and the error compensated by the system with $k = 0.25$. The same data series is used as for figure 3.9. One can see that the peak at 0.66 Hz is not totally suppressed anymore. However, the remaining of the frequency range is less amplified than by the system with $k = 1$ (compare with figure 3.9). Still, the error power is not reduced, only the powerspectrum is flattened.

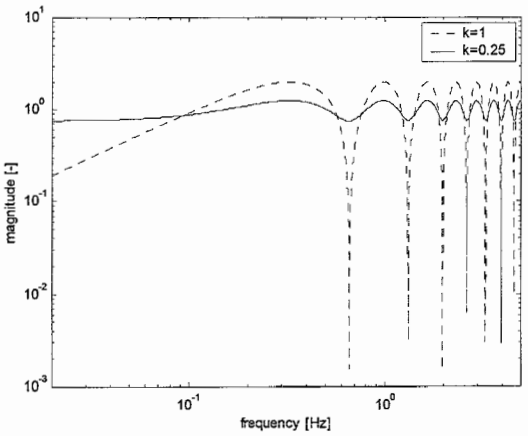

Figure 3.11: Magnitude plots for different gains

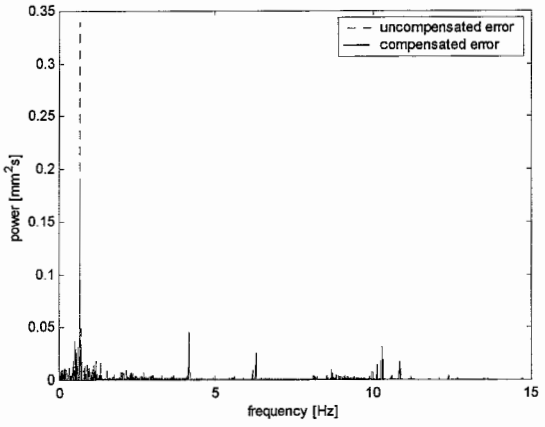

Figure 3.12: Powerspectrum of error signal

In order to improve the system's performance, a second modification is made. In the powerspectrum one can see that the fundamental frequency 0.66 Hz hardly has multiples. However, the designed system does suppress these multiples and amplifies the frequencies between them. In order to prevent this, a low-pass filter *q(z)* can be added as shown in figure 3.13. The low-pass filter will weaken and finally cancel the effect of the memory loop above a certain cut-off frequency. The best result will be obtained if the low-pass filter has a sharp cut-off at 0.66 Hz. This way the memory loop will only suppress the most powerful frequency and it will not deal with higher frequencies.

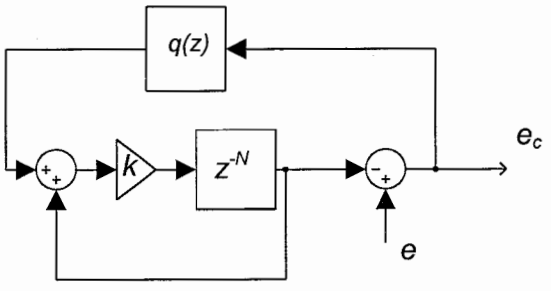

Figure 3.13: Configuration with additional gain and low-pass filter

The chosen low-pass filter is one of the FIR filter type (see chapter 4 for explanation). In order to obtain a sharp cut-off, a high order filter must be chosen, especially because the sampling frequency (500 Hz) is high in comparison with the cut-off frequency (0.66 Hz). Figure 3.14 shows the magnitude plot of a 100th order low-pass FIR filter with a cut-off frequency as low as possible (below a certain frequency, further decrease of the cut-off frequency has no influence). As can be seen, only frequencies above 10 Hz are fully suppressed. The transfer function of the new configuration can be written as:

$$H = \frac{z^N - k}{z^N - k + q(z)k} \tag{3.12}$$

Again an optimal gain can be searched for. In this case the optimal gain *k* is found to be approximately 0.45. Figure 3.15 shows the magnitude plot of this transfer function together with the magnitude plot of the transfer function without the low-pass filter

(3.11). It can be seen that the magnitude of the system including the low-pass filter approaches unity above 10 Hz.

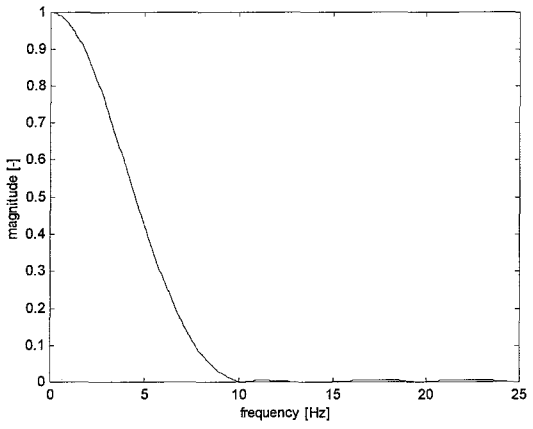

Figure 3.14: Magnitude of low-pass filter

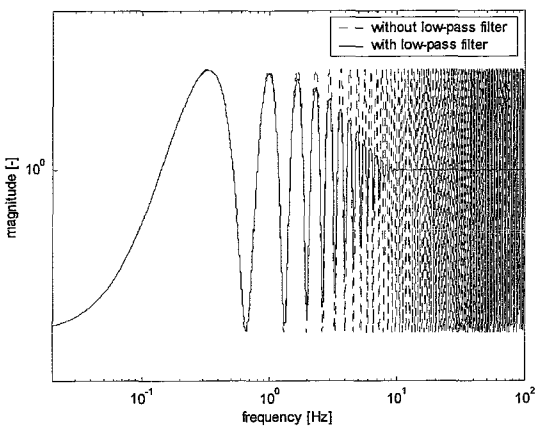

Figure 3.15: Magnitude of system with low-pass filter

Simulations are repeated for the new system. The results show a decrease of error power. On average the compensated error power is about 15% lower than the uncompensated error power.

In principle this system can be used. However, a very high order low-pass filter is needed in order to obtain satisfactory results. The high order of the filter can cause problems in the real implementation. A solution to these problems is decreasing the sampling frequency of the system, because the order of the low-pass filter will decrease proportionally with the sampling frequency. The choice is made to decrease the sampling frequency with a factor 10. This way only a 10^{th} order instead of a 100^{th} order low-pass filter is needed to give the same magnitude characteristic. However, the sampling frequency of the measurements is still 500 Hz. In order to adapt the input signal to the system's sampling frequency, the measured signal is averaged over 10 samples each time. Furthermore, the compensation signal must be available with a frequency of 500 Hz, so the 50 Hz output signal of the system must be changed back to a 500 Hz signal. Interpolating the 50 Hz output signal accomplishes this. Simulations show that the decrease of the system's sampling frequency does not influence its performance. The explanation for this is that a sampling frequency of 50 Hz is still large enough to describe the compensation signal of 0.66 Hz correctly.

Instead of decreasing the order of the low-pass filter proportionally with the sampling frequency, it can be kept constant as well. This way the magnitude characteristic of the filter shows a sharper cut-off (figure 3.16), which results in a magnitude plot of the system that already approaches unity at 1 Hz instead of 10 Hz (figure 3.17). Simulations performed with the corresponding optimal gain show that this modification results in an extra decrease of error power of a few percents. However, the problem of the high order filter is still present. Reducing the sampling frequency of the system again could solve the problem. However, a too low sampling frequency will result in a poor description of the compensation signal, which yields an increase of error power.

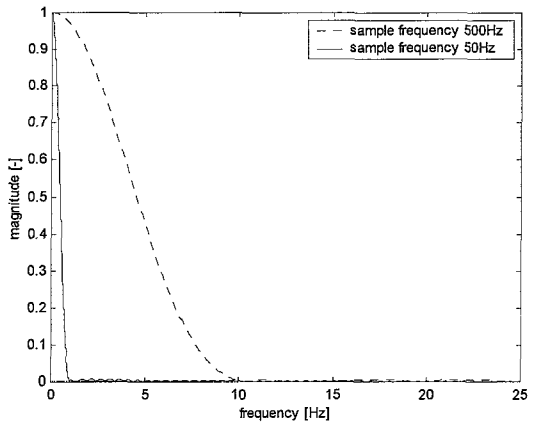

Figure 3.16: Magnitude of low-pass filters

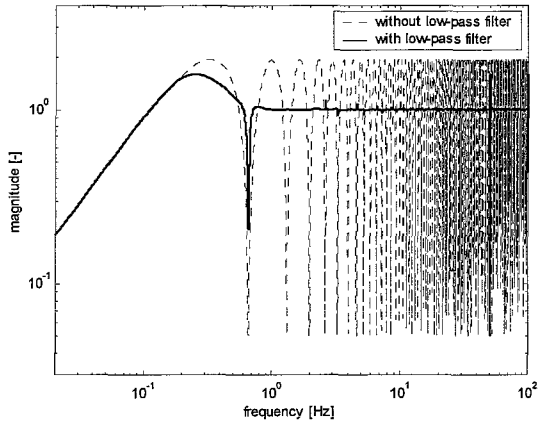

Figure 3.17: Magnitude of system with low-pass filter

It's clear that a trade-off must be made in order to tune the system's parameters. It can be summarised as follows: The low-pass filter must have a cut-off as sharp as possible. This is achieved by either reducing the sampling frequency or increasing the filter order. On the one hand reducing the sampling frequency could result in a worse description of the compensation signal. On the other hand, increasing the filter order can give problems in the real implementation. Trial and error will yield the best system parameters.

The simulations performed so far predicted (a part of) the error only one sample before it was measured. However, in the real implementation the prediction and compensation of the error must occur approximately one second in advance. Simulations are performed in order to determine whether this influences the results. It turns out that this delay does not influence the results significantly.

4. Active Noise Control

Active noise control (ANC) is a signal processing strategy in which additional secondary sources are used to cancel noise from the original primary source. It is based on the principle of superposition: an anti-noise of equal amplitude and opposite phase is generated and combined with the primary noise, resulting in cancellation of both noises.

Two types of noise can be distinguished: broadband and narrowband. Broadband noise is totally random, which means that its power is (approximately) equally distributed across the frequency spectrum. To cancel this kind of noise broadband feedforward ANC is used. This technique uses a reference sensor close to the noise source to measure a signal that gives an indication of the approaching noise. Primary noise that correlates with the reference signal will be cancelled downstream with a secondary source. Narrowband noise concentrates most of its power at specific frequencies. A technique that is effective in reducing this repetitive noise is called narrowband feedforward ANC. This technique does not use a reference sensor to describe the approaching noise, but uses information about the frequencies of the noise signal instead. This information can be obtained offline in case of a stationary noise signal or online in case of a time varying noise signal, for example by a tachometer. If there is no reference sensor or other a priori knowledge of the primary noise signal, feedback ANC is used. In this approach only the resulting error is fed back in order to cancel the primary noise.

4.1 Narrowband feedforward ANC

The problem of compensating the position error of the OPC belt is a typical problem for narrowband feedforward ANC. The condition that most of the power of the signal is concentrated at specific frequencies is satisfied, because half the signal power is present at only five frequencies (see chapter 2). Furthermore, these frequencies are known in advance.

Figure 4.1 shows the configuration of a narrowband feedforward ANC system. An ANC system consists of two main parts: a digital filter and an adaptive algorithm. The digital filter processes the incoming reference signal r and generates the secondary signal y that compensates the primary noise signal e. The adaptive algorithm adapts the filter coefficients at runtime in order to minimise the mean square value of the output e_c.

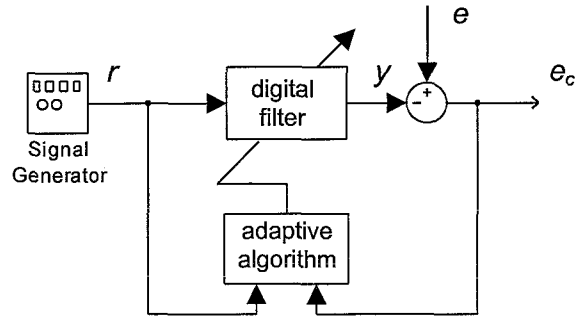

Figure 4.1: Narrowband feedforward ANC configuration

Usually a Finite Impulse Response (FIR) filter is used, which is schematically depicted in figure 4.2.

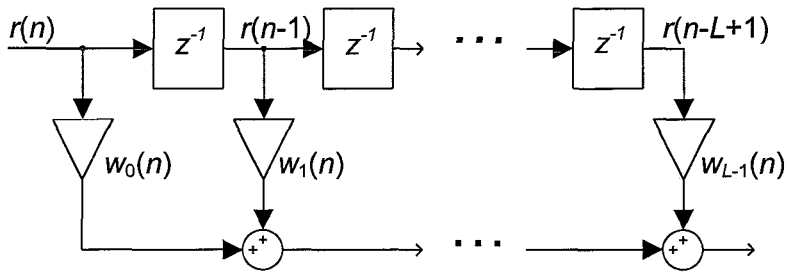

Figure 4.2: Block diagram of FIR filter

Given a set of L filter coefficients, $w_l(n)$, $l = 0, 1, \ldots, L-1$, and a data sequence $\{r(n)\ r(n-1)\ \ldots\ r(n-L+1)\}$ (n denotes the sample number), the output signal is computed as:

$$y(n) = \sum_{l=0}^{L-1} w_l(n) r(n-l) \tag{4.1}$$

If the reference signal vector and weight vector are respectively defined as:

$$\underline{r}(n) \equiv [r(n) \quad r(n-1) \quad \ldots \quad r(n-L+1)]^T \tag{4.2}$$

$$\underline{w}(n) \equiv [w_0(n) \quad w_1(n) \quad \ldots \quad w_{L-1}(n)]^T \tag{4.3}$$

Then the output signal $y(n)$ can be written as:

$$y(n) = \underline{w}^T(n)\underline{r}(n) \tag{4.4}$$

The compensated error signal $e_c(n)$ can now be written as:

$$e_c(n) = e(n) - y(n) = e(n) - \underline{w}^T(n)\underline{r}(n) \tag{4.5}$$

4.2 Adaptive algorithm

The adaptive algorithm updates the coefficients of the digital filter to optimise some predetermined performance criterion. Usually the mean square error (MSE) is minimised, which is defined as:

$$\xi(n) \equiv E[e_c^{\,2}(n)] \tag{4.6}$$

where $E[\,]$ is the expectation operator. Substitution of equation (4.5) in (4.6) results in the following expression for the MSE:

$$\xi(n) \equiv E[e^2(n)] - 2\underline{p}^T \underline{w}(n) + \underline{w}^T(n) R \underline{w}(n) \tag{4.7}$$

where $\underline{p}$ is the cross-correlation vector from the input error $e(n)$ to the reference signal $r(n)$ and R is the auto-correlation matrix of the reference signal $r(n)$.

$$\underline{p} \equiv E[e(n)\underline{r}(n)] \tag{4.8}$$

$$R \equiv E[\underline{r}(n)\underline{r}^T(n)] \tag{4.9}$$

The MSE is a function of the filter coefficients $\underline{w}(n)$. For each value of the filter coefficient vector $\underline{w}(n)$, there is a corresponding value of the MSE. The MSE values associated with a filter of L weights thus span an $(L+1)$ dimensional space that is called the performance surface. For $L = 2$ this can be presented graphically as is shown in figure 4.3. The values $w_0^o(n)$ and $w_1^o(n)$ are the optimal coefficients of the digital filter, which result in a minimum MSE.

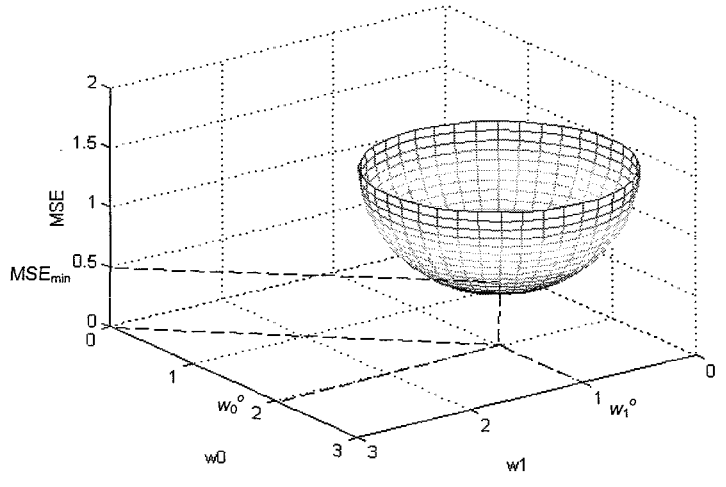

Figure 4.3: Performance surface

The optimal filter coefficient vector $\underline{w}^o(n)$ can be found by differentiating the expression for the MSE (equation 4.7) with respect to the filter coefficient vector $\underline{w}(n)$ and nullifying the result:

$$-2\underline{p} + 2R\underline{w}^o(n) = 0 \tag{4.10}$$

Calculating the solution will generally demand a great computational effort. Therefore, a recursive method for computing $\underline{w}^o(n)$ is used. To minimise the MSE one needs to descend along the performance surface by adjusting the filter coefficients until the minimum MSE is reached. The steepest descent method is a method that reaches the minimum by following the direction in which the performance surface has the greatest rate of decrease. This means that the weights are updated in the direction of the negative gradient of the performance surface:

$$\underline{w}(n+1) = \underline{w}(n) - \frac{\mu}{2}\nabla\xi(n) \tag{4.11}$$

where μ is a convergence factor that controls stability as well as the rate of convergence to the optimal filter coefficients. The vector $\nabla\xi(n)$ is the gradient of the MSE with respect to the filter coefficient vector $\underline{w}(n)$:

$$\nabla\xi(n) = -2\underline{p} + 2R\underline{w}(n) \tag{4.12}$$

Substitution of equation (4.12) in (4.11) results in the final form of the algorithm:

$$\underline{w}(n+1) = \underline{w}(n) + \mu[\underline{p} - R\underline{w}(n)] \tag{4.13}$$

The weight vector $\underline{w}(n)$ is adapted until it has converged to its optimum $\underline{w}^o(n)$ so that the gradient becomes zero.

The method of steepest descent requires the statistics of $e(n)$ and $r(n)$ to be known. However, in many practical applications these are unknown so the method cannot be used directly. As an alternative the instantaneous squared compensated error e_c^2 can be used to estimate the mean square error defined in equation (4.6):

$$\hat{\xi}(n) = e_c^{\,2}(n) \tag{4.14}$$

It follows that the gradient of the MSE is estimated by:

$$\nabla\hat{\xi}(n) = 2[\nabla e_c(n)]e_c(n) \tag{4.15}$$

Substitution of equation (4.5) results in the following expression for the gradient estimate:

$$\nabla\hat{\xi}(n) = -2\underline{r}(n)e_c(n) \tag{4.16}$$

Substituting this in the equation of the steepest descent method (4.11) gives:

$$\underline{w}(n+1) = \underline{w}(n) + \mu\underline{r}(n)e_c(n) \tag{4.17}$$

The above equation is called the least mean square (LMS) algorithm.

The convergence rate μ must be selected in order to obtain stability and convergence. A sufficient but not necessary condition for stability is derived in [3]:

$$0 < \mu < \frac{2}{LP_r} \tag{4.18}$$

where P_r is the signal power of the reference signal $r(n)$.

4.3 Single frequency ANC

If a sine wave is applied as the reference input, the LMS algorithm becomes an adaptive notch filter. It removes certain frequency components within a small band around the reference frequency. The adaptive notch filter is especially useful when the interfering sinusoid drifts slowly in frequency. Figure 4.4 shows a single frequency ANC system.

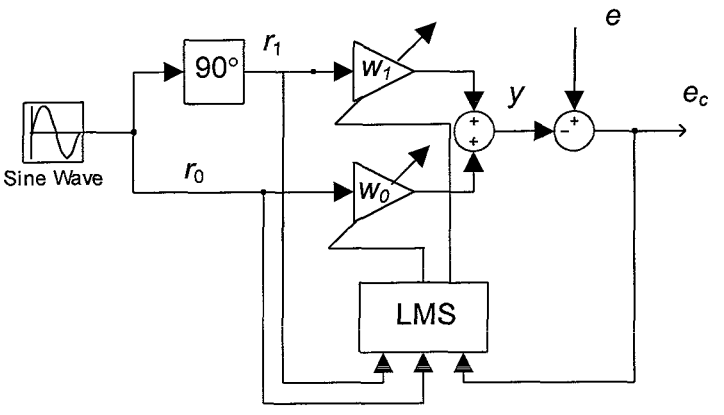

Figure 4.4: Single frequency ANC system

The primary noise signal $e(n)$ is assumed to be a sine wave of the following form:

$$e(n) = A_e \sin(\omega_0 n + \phi_e) \tag{4.19}$$

where A_e and ϕ_e are respectively the (unknown) amplitude and phase of the primary noise signal. ω_0 is the frequency [rad/sample] of the signal, which must be known. Because the primary noise signal is a sine wave, the reference-input $r_0(n)$ is chosen to be a sine wave of equal frequency:

$$r_0(n) = A\sin(\omega_0 n) \tag{4.20}$$

By shifting the input signal 90° a cosine wave $r_1(n)$ is generated:

$$r_1(n) = A\cos(\omega_0 n) \tag{4.21}$$

Correct linear combination of the two waves yields a secondary signal $y(n)$ that is equal to the primary noise signal $e(n)$.

The LMS algorithm for this system is expressed as follows:

$$y(n) = w_0(n)r_0(n) + w_1(n)r_1(n) \tag{4.22}$$

$$e_c(n) = e(n) - y(n) \tag{4.23}$$

$$w_l(n+1) = w_l(n) + \mu r_l(n)e_c(n), \quad l = 0,1 \tag{4.24}$$

Note that equation (4.22) can be obtained from equation (4.4) by taking:

$$\underline{r}(n) \equiv [r_0(n) \quad r_1(n)]^T \tag{4.25}$$

4.4 Multiple frequency ANC

A single frequency component can be cancelled by a two weight adaptive filter. The cancellation of M frequency components can be achieved by connecting M two weight adaptive filters in parallel. The secondary signal $y(n)$ is a sum of M individual compensation signals:

$$y(n) = \sum_{m=1}^{M} y^m(n) \tag{4.26}$$

The multiple frequency ANC system is depicted in figure 4.5 for $M = 2$ frequencies.

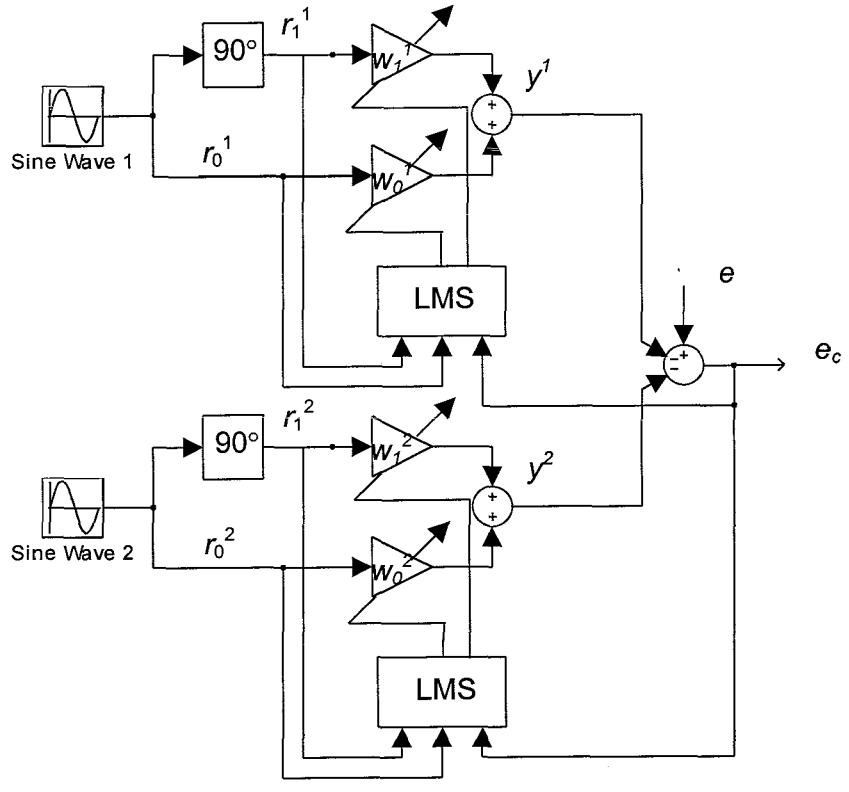

Figure 4.5: Multiple frequency ANC system

4.5 Application of ANC

The ANC signal processing strategy is used to predict the position error of the OPC belt. First single frequency ANC is used to compensate the most powerful frequency in the error signal: 0.66 Hz. Before simulations can be performed, the algorithm's variables must be initialised. In this case the initial values of the filter weights and the convergence rate μ must be selected.

Convergence and initial performance depend on the initial values of the filter weights. Because the phase of the disturbance signal is unknown, no well-founded initial guess of the filter weights can be made. Therefore, the filter weights are given a low initial value of 0.001.

Selection of the convergence rate μ can be started by fulfilling the stability condition (4.18). To do this, the reference signal power P_r must be computed, which is defined as:

$$P_r = E[r^2(n)] \tag{4.27}$$

The reference signal $r(n)$ is a sine wave with unit amplitude. This results in $P_r = 0.5$ mm^2. The single frequency ANC system uses two weights, so $L = 2$. Substitution of these values in equation (4.18) gives an upper bound on the convergence rate of 2. In order to obtain maximum convergence, the convergence rate should be chosen as

large as possible, which is in this case 2. However, the prediction and compensation of the error must occur approximately 1 second before the error occurs. Therefore, 1-second-old filter weights are used to predict the error signal. A large convergence rate changes the filter weights very rapidly, so filter weights which are 1 second old will deviate a lot from the current filter weights. As a result the compensation signal will be bad and performance will be poor. Figures 4.6 and 4.7 show the results of simulations with convergence rate of 2 and 0.0003 respectively. Figures a show the uncompensated error signal together with the compensation signal and the compensated error signal. In these figures the compensation signal is generated using the current filter weights. Figures b show the filter weights. Figures c show the uncompensated error signal together with the compensation signal and the compensated error signal as well. However, in these figures the signals are 1 second ahead and compensated with the same filter weights as in figures a.

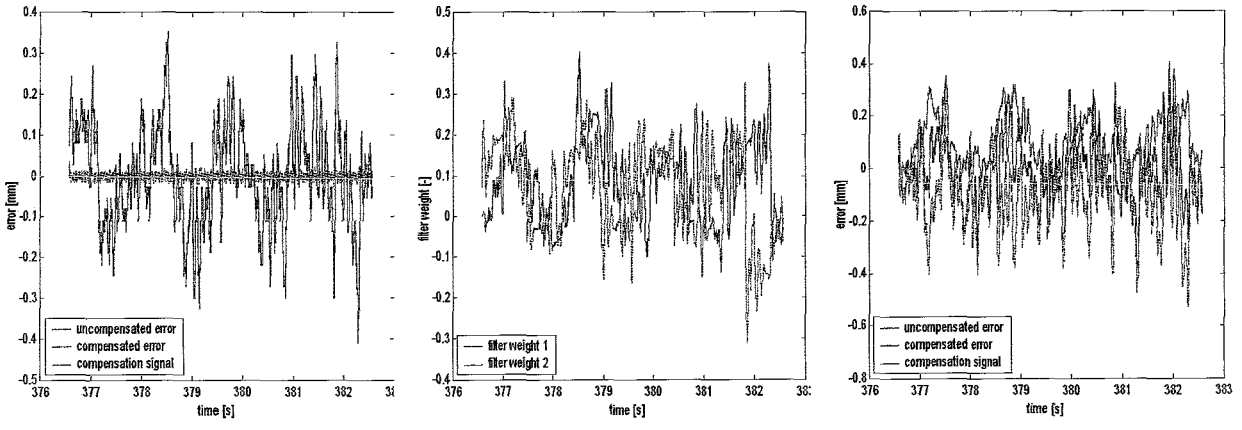

Figure 4.6a/b/c: Simulation results with large convergence rate

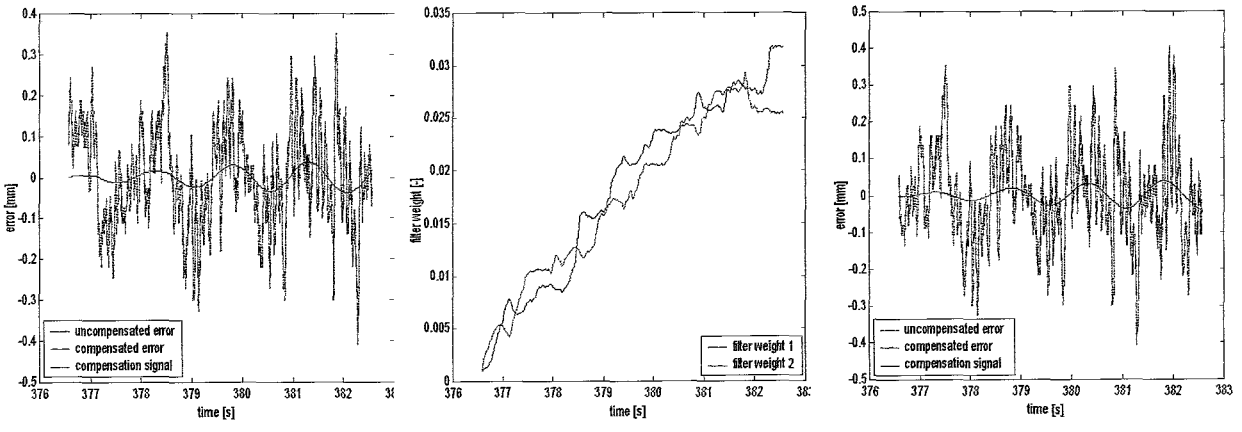

Figure 4.7a/b/c: Simulation results with small convergence rate

Figure 4.6a shows that a large convergence rate results in fast convergence and good instantaneous compensation of the error signal. However the filter weights change very rapidly (figure 4.6b) and are therefore no longer valid after 1 second. This is visible in figure 4.6c: the compensated error is even larger than the uncompensated error. Figure 4.7a shows that a small convergence rate results in slow convergence. However, the compensation signal has the shape of a sine wave and slowly increases in amplitude. In steady state this sinusoidal part of the error signal will be compensated. Figure 4.7b shows that the filter weights change a lot

slower than in figure 4.6b. This results in a better compensation with 1-second-old filter weights, as can be seen in figure 4.6c. It is clear that a trade off must be made between convergence speed and steady state error in order to find the best result.

To improve convergence speed and maintain small steady state error, the convergence rate μ is made variable. Initially the convergence rate is large in order to obtain fast convergence. After the algorithm has converged, the convergence rate is decreased until a final minimum convergence rate is reached. A way of updating the convergence rate is discussed in [4]. The value of μ is decreased by a factor α if $r(n-1)e(n)$ alternates sign on m successive samples. The parameters m and α are usually chosen to be 3 and 2 respectively. However, simulations show that $\alpha = 1.5$ provides better results in this particular case. The initial and final convergence rate must be selected as well. Simulations show that an initial convergence rate of 0.03 and a final convergence rate of 0.0003 provide best results.

Figure 4.8 shows the same results as figures 4.6 and 4.7. However, the results were obtained using the variable convergence rate algorithm. In figure 4.8b an extra line is visible, which represents the convergence rate. One can see that initially the filter weights change very rapidly (figure 4.8b), resulting in fast convergence (figure 4.8a). After some time the convergence rate decreases, resulting in slower changing of the filter weights. The compensation signal then becomes a steady sine wave.

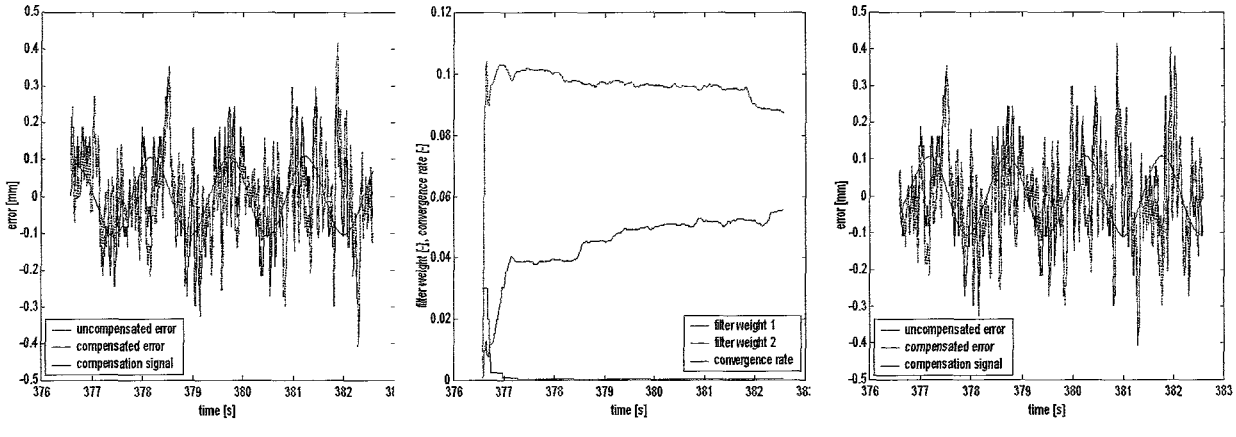

Figure 4.8a/b/c: Simulation results with variable convergence rate

Simulations are performed with the variable convergence rate algorithm for all ten available data series. The power of the uncompensated error is compared with the power of the compensated error using one second old filter weights. In the first six seconds of a run, the error power decreases approximately 11% on average. It must be noted that the standard deviation of this result is 10%, indicating that the results of individual simulations vary a lot.

In order to try to improve performance, the single frequency ANC system is extended to a multiple frequency ANC system. First the system is adjusted so that the two most powerful frequencies (0.66 Hz and 4.15 Hz) can be compensated. Also the parameters for the 4.15-Hz-part must be selected. Simulations show that an initial convergence rate of 0.01 provides best results. The remaining parameters are chosen equal to the parameters of the 0.66-Hz-part. In the first six seconds of a run, this new configuration gives an average error power decrease of approximately 17% in comparison with the uncompensated error power. The standard deviation of this result amounts to 18%.

Next the system is extended with a part for compensating a third frequency: 10.29 Hz. After optimising the algorithm's parameters, the error power reduction is

determined again. On average the error power decreases with approximately 11% in comparison with the uncompensated error power, with a standard deviation of 20%. This means that adding the third frequency part worsens the result. The cause of this can be that the power at 10.29 Hz is too low for the algorithm to converge well.

To determine how the algorithm performs after it has converged, simulations of one minute length are performed. These simulations are performed with the two frequencies ANC system. In the time range from six to sixty seconds an average error power reduction of approximately 26% with a standard deviation of 14% is obtained. In order to visualise this result, a histogram of the error amplitudes is included in appendix a.

Finally the robustness for frequency variations is investigated. This is done by increasing all reference signal frequencies with one percent. Simulations are repeated with the two frequencies ANC system on all available data series. On average the error power decreases with approximately 4% in comparison with the uncompensated error power, with a standard deviation of 22%. This result shows that the performance of the algorithm is sensitive to changes in frequency.

The high standard deviations show that the results vary a lot. The deviation is mainly the effect of the large deviation of the individual data series. Furthermore, the initial convergence plays a role in performance. A coincidental bad initial convergence can lead to bad results. The large deviation makes it very difficult to determine optimal system parameters. However, a few general remarks can be made about the optimisation of the system parameters. First, if the error signal contains a great part of random noise, the convergence rate should be chosen low. This will prevent the system from converging to the random part instead of the repetitive part of the error signal. Second, only frequencies that show a clear peak in the powerspectrum should tried to be cancelled. Otherwise, the system will not be able to converge well, resulting in poor performance. Third, a variable convergence rate improves convergence speed and maintains small steady state error and is therefore advisable for non-stationary signals.

5. Comparison of repetitive control with active noise control

In the past two chapters, two strategies for predicting and compensating repetitive errors have been discussed. Both strategies are non-model based, only a repetitive part of the error signal is compensated. The strategies are also similar in a way that the frequency of the repetitive errors must be known in advance. Each strategy has its own advantages and disadvantages. The strategies are here compared on several general points as well as suitability for this specific problem. The comparison will result in a choice for a strategy to be implemented.

Chapter 3 discusses the repetitive control strategy. This strategy is especially useful for compensating frequencies with higher harmonics, which are very often present in mechanical vibrations. Repetitive control uses a memory loop to generate a compensation signal, so any arbitrary repetitive error signal with a fixed frequency can be compensated. The side effect of this memory loop is that frequencies between multiples of the fundamental frequency are amplified. A repetitive control system has fully converged after one period of the fundamental frequency. In order to cancel out the effect of the memory loop at higher frequencies, a low-pass filter can be added. If only one frequency must be cancelled, a very high order low-pass filter is needed, especially if the sampling frequency is high in comparison with the error frequency. Another effect of a high sampling frequency in comparison with the error frequency is a large memory loop and as a result a large data-storage capacity is required.

Active noise control is discussed in chapter 4. This strategy uses a digital filter to adapt a reference signal in order to compensate (a part) of the error signal. An adaptive algorithm adapts the filter weights. Usually a sine wave is used as a reference signal and as a result only error signals with a shape similar to a sine wave can be compensated. An ANC system can be extended in order to compensate more frequencies. These frequencies need not be multiples of each other. The convergence speed of an ANC system depends on the chosen convergence rate as well as the amount of noise of other frequencies that is present in the error signal. The differences of the two strategies are summarised in the table below.

Repetitive control	Active noise control
Any arbitrary repetitive error signal with fixed frequency can be compensated	Only (parts of) error signals that are (similar to) a sine wave can be compensated
Full convergence after one period	Convergence speed depends on parameter settings and the amount of noise of other frequencies
Suppression of multiples of fundamental frequencies as well	No suppression of multiples of fundamental frequency. However, possibility to extend system to compensate more frequencies
Amplification of frequencies between multiples of the fundamental frequency	No amplification of other frequencies
Computational effort and required data-storage capacity can be high due to low-pass filter and memory loop respectively	Low computational effort and hardly no data-storage capacity needed

Table 5.1: Comparison of repetitive control with active noise control

Simulations performed on error data obtained from the experimental set-up show that more error power reduction can be obtained using the ANC strategy than using the repetitive control strategy. Furthermore, the computational effort and

memory storage for the ANC system is much lower than for the designed repetitive control system. The ANC system is also easier to implement than the repetitive control system. For these reasons, the choice is made to implement the ANC strategy in the experimental set-up.

6. Implementation

To determine whether the designed ANC system improves the accuracy of registration significantly, the algorithm must be tested on the experimental set-up. To realise this, the algorithm has to be implemented in the C language. This is done by first translating the Simulink model to a few Matlab functions and then translating these functions to the C language.

6.1 Composition of the functions

The designed ANC system can be captured in three functions: an initialisation function, a function to update the filter coefficients and a function to predict the error of the OPC belt. The initialisation function assigns the initial values to the variables. This function must be run at the start up of a print session and has no input value. The update function receives the measured error from the encoder every 2 ms and updates the filter weights in order to minimise the mean square error. The prediction function is run each time a new sheet of paper must be printed. It uses the current filter weights to predict what the position error of the OPC belt will be by the time the image is transferred to the TTF belt. The input of this function is the amount of time to the moment of transfer, which was earlier estimated at one second. The outcome of the function is used to correct the time at which the image is printed onto the OPC belt. This results in compensation of the error. The described Matlab functions are included in appendix B.

Next the Matlab functions are translated to the C language. There are a few restrictions when translating the functions. The algorithm must run very fast on a processor that is as cheap as possible (calculations are performed every 2 ms). Therefore sine and cosine functions are not allowed. These functions must be approached with tables. Furthermore, variables of type 'float' take a great computational effort and are therefore not allowed either. This means that only integers can be used. In order to preserve accuracy, decimal numbers must be scaled. This can be achieved by multiplying them with a large number. An integer can take any whole number between –32767 and 32767, so scaling must be done in such a way that the resulting number does not exceed these limits. However, the resulting number must contain enough digits in order to maintain accuracy. These restrictions can cause problems if the variable to be scaled varies in a great range and if great accuracy is required as well.

The final form of the C program is tested on the experimentally obtained error data and its results are compared with the results obtained using the Matlab functions. It appears that the results of the C program only slightly differ from the results obtained using the Matlab functions. These small differences can be attributed to loss of accuracy due to the scaling and approximation of the (co)sine functions. Appendix C shows the final C program. The written C functions are integrated with the total software that runs on the processor in order to be able to perform experiments.

6.2 Experimental results

First, experiments are performed without using the prediction function. Only the initialisation function and update function are run. During these experiments some data are stored in order to determine whether the algorithm converges well. Several data series of twenty seconds length are measured. The results show that the algorithm converges in a way similar to the offline simulations. However, some of the new powerspectra of the uncompensated error show high peaks at 1.31 Hz that have not been seen before. It appears that these peaks slowly appear and disappear

during a run. These new peaks are probably the effect of mechanical and control changes that were made to the experimental set-up after the original powerspectra had been measured. Figure 6.1 shows the powerspectra of two data series of twenty seconds length obtained at different times during a single run. It can be seen that figure 6.1a does not show a peak at 1.31 Hz and that figure 6.1b does.

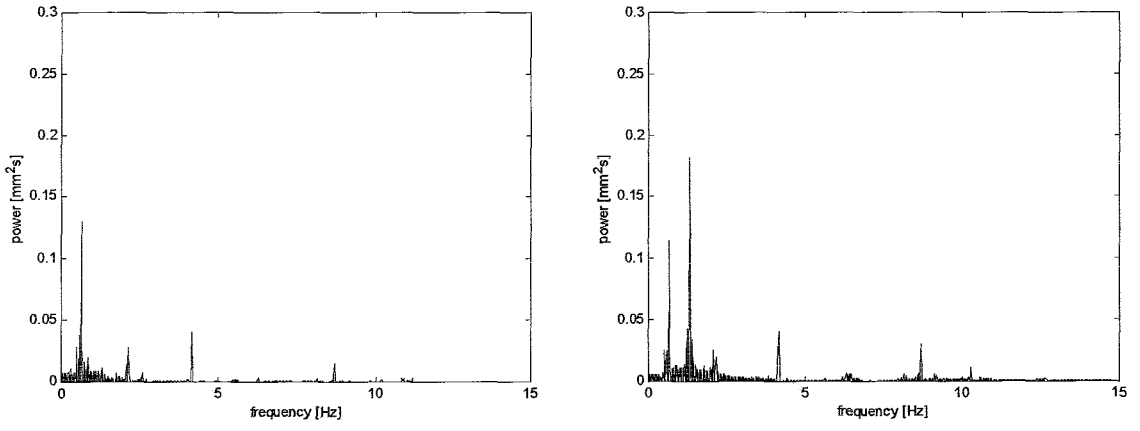

Figure 6.1a/b: Newpowerspectra obtained during a single run

Because the peak at 1.31 Hz contains a great a part of the error power, it is decided to compensate this frequency instead of 4.15 Hz which has a much lower peak. The experiments are repeated for the new situation. Several data series are measured from the start of a run, in order to determine how the algorithm converges. Also, data series are measured some time after the start in order to determine the steady state performance. On average the instantaneous error power reduction of these new simulations is approximately 25% for the series that are measured from the start. The average instantaneous error power reduction of the steady state measurements is approximately 60%. Figure 6.2 shows the results of a data series that is measured in steady state. In figure 6.2a the uncompensated error signal is visible together with the instantaneous compensated error signal as well as the compensation signals. One can see that the compensation signals converge to rather steady sine waves and that a significant reduction in error amplitudes is accomplished. Figure 6.2b shows that the filter weights only change slowly, which means that they are still valid after one second. However, it can be seen that the filter weights of the 1.31 Hz signal change faster than the filter weights of the 0.66 Hz signal. Possible explanations for this phenomenon can be the appearance and disappearance of the 1.31 Hz component during a run or the disturbance frequency not being exactly equal to 1.31 Hz.

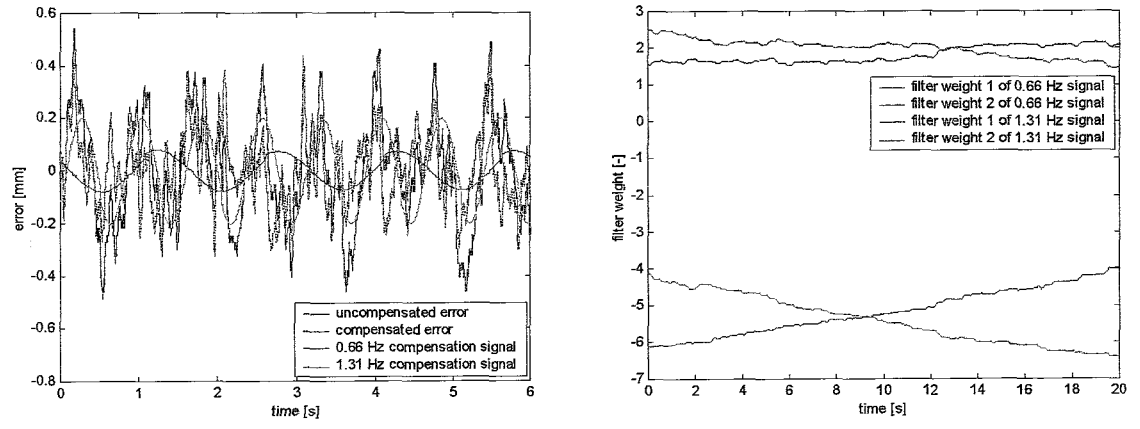

Figure 6.2a/b: Experimental results of algorithm in steady state

The experiments performed so far show that online the algorithm behaves in a way that is similar to the offline simulations. The next step is to include the prediction function in order to adapt the start of page moment. Experiments are performed in which the prediction function is run each time that a new sheet has to be printed. The predicted error is compared with the real error that occurs. Figure 6.3 shows the real error and predicted error as well as the difference between the two. Figure 6.3a shows the results of the start of a run, whereas figure 6.3b shows the results of the algorithm in steady state. One can see that the prediction function indeed predicts a part of the error signal.

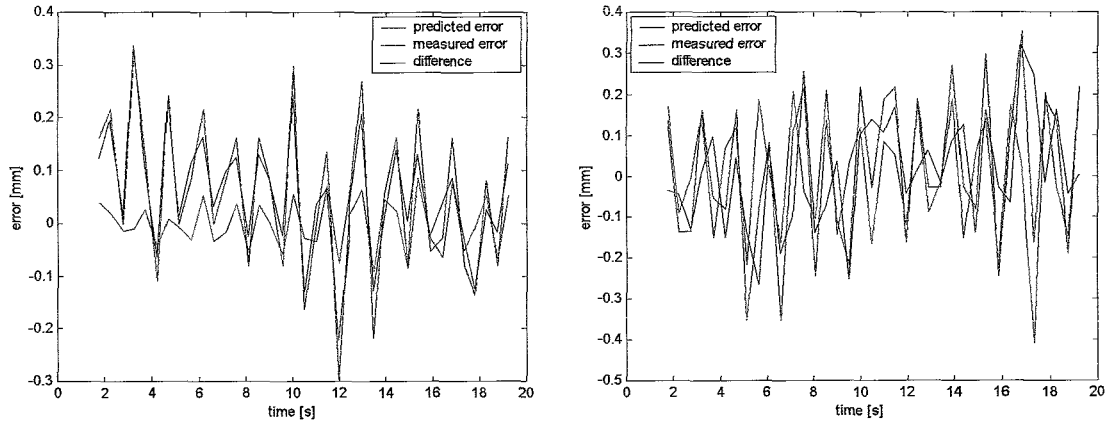

Figure 6.3a/b: Predicted error compared with measured error

Finally, experiments are performed in which the predicted error is used to adjust the start of page moment. Two print sessions of 1500 sheets each are run. One session is run using ANC to adjust the start of page moment and one session is run without adjusting the start of page moment. Of each run, approximately 150 sheets are selected and are scanned in order to analyse the registration accuracy. These 150 sheets are equally distributed over a run. The reason why 150 sheets are selected out of a total of 1500 sheets, instead of printing 150 sheets in the first place, is to cancel out the effect of low frequent changes in disturbance level. Earlier performed experiments proved the existence of these changes. The results of the two print sessions are compared. It appears that no significant change in registration accuracy is made using ANC, although the algorithm performs well, as can be concluded from the previous figures. The explanation for this is probably that the experimental set-up produces too many other disturbances that affect the registration accuracy. However, by the time this experimental set-up will be ready for production, these other disturbances will be smaller due to improved mechanics and control strategies. The experiments should then be repeated in order to determine whether a significant improvement in registration accuracy can be made using ANC.

Conclusions and recommendations

The goal of this research was to design an algorithm that predicts the position error of the OPC belt, so that it can be compensated by adjusting the time at which the image is printed onto the belt. In order to realise this, first some experimentally obtained error data have been analysed. Next, two control strategies that are able to counteract periodic disturbances have been discussed: repetitive control and active noise control. Both strategies have been adapted to the problem and have been optimised by means of simulations in order to obtain maximum error power reduction. The two strategies have been compared on their suitability for solving the problem, resulting in a choice for a strategy to be implemented in the experimental set-up. In order to implement the chosen algorithm, it has been translated to the C language. Finally, experiments have been performed in order to determine whether the algorithm performs well online and whether a significant improvement in registration accuracy is made.

Offline analysis of the experimentally obtained error data shows that a significant part of the error power is present at only a few fixed frequencies. If the most powerful frequency is filtered out by averaging, an average error power reduction of approximately 25% is obtained. If this is done for the five most powerful frequencies an average error power reduction of approximately 50% is obtained.

Repetitive control and active noise control have been used to predict and compensate the position error of the OPC belt. Simulations performed on the experimentally obtained error data show that the largest error power reduction can be obtained using active noise control. In steady state, this error power reduction is approximately 25%. Besides the larger error power reduction, the active noise control strategy requires less computational effort and is easier to implement than the designed repetitive control system. For these reasons, the choice has been made to implement and test this strategy on the experimental set-up.

Experiments performed on the experimental set-up show that the online performance of the algorithm is similar to the offline performance, which means that the algorithm performs well. However, using the algorithm to adapt the moment of start of page does not change the registration accuracy significantly. This is probably the effect of the existence of many other disturbances in the experimental set-up that influence registration accuracy. Nevertheless, in a later stage of the development these disturbances will be smaller due to improved mechanics and control strategies. In combination with the fact that earlier performed experiments show that the position error of the OPC belt is directly related to the registration accuracy, it is expected that the registration accuracy will then significantly improve by using active noise control.

It is recommended to repeat the experiments on other experimental set-ups in order to test the robustness of the algorithm. The frequencies to be compensated must be given a value offline, so the algorithm will only perform if the error frequencies of these set-ups are equal to the error frequencies of the first experimental set-up. It is expected that these frequencies are indeed equal, because the frequencies are mainly a function of the velocity of the OPC belt. This velocity will be the same in all set-ups. However, simulations show that the performance of the algorithm strongly depends on the accuracy with which the error frequencies have been estimated. Therefore, the algorithm will probably perform better on different set-ups if the error frequencies are determined online.

The algorithm has to converge each time a new print session is started. However, for a certain reason the transfer pinch sometimes has to be opened during a run (see figure 1.2 for transfer pinch). This results in an almost instantaneous extra displacement of the OPC belt. The effect of this is that the filter weights must converge again. It is therefore recommended to restore the convergence rate to its initial high value when the transfer pinch opens.

Appendix A: Histogram error signal

Figure A.1 shows a histogram of the amplitudes of a compensated and an uncompensated error signal. The compensated signal is obtained using the two frequencies ANC system with variable convergence rate. Both signals are of one minute length. It is visible that the compensated error signal has smaller amplitudes than the uncompensated error signal.

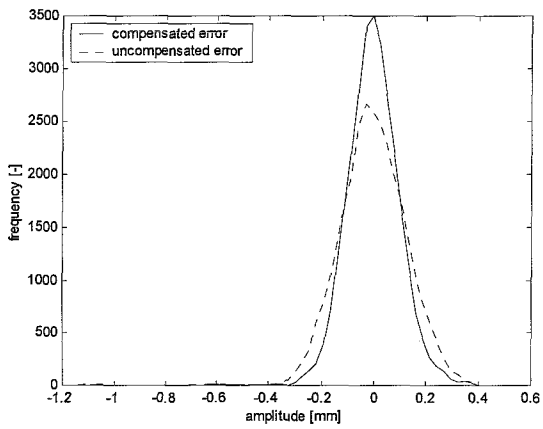

Figure A.1: Histogram error amplitudes

Appendix B: Matlab functions

Initialisation function

```
clear f1 f2 mu1 mu2 ec ecv ecvv w1 w2 ww1 ww2  alf i er  f1t f2t
global f1 f2 mu1 mu2 ec ecv ecvv w1 w2 ww1 ww2  alf i er f1t f2t
i=1;         %sample counter
fs=500;      %sampling frequency [sample/s]
f1t=0.66;    %frequency 1 [periods/s]
f1=f1t/fs;   %frequency 1 [periods/sample]
f2t=4.151;   %frequency 2 [periods/s]
f2=f2t/fs;   %frequency 2 [periods/sample]
mu1=0.03;    %initial convergence rate 1
mu2=0.01;    %initial convergence rate 2
ec=0.001;    %initial value corrected error
ecv=0;       %initial value corrected error one sample earlier
ecvv=0;      %initial value corrected error two samples earlier
y=0;         %initial correction signal
w1=[0.001; 0.001]; w2=[0.001; 0.001];  %initial filter weights
```

Update function

```
function ec=updatefunc(e)            %input is measured error
global f1 f2 mu1 mu2 ec ecv ecvv w1 w2 ww1 ww2 alf i er

%first frequency
r1=[sin(2*pi*f1*i);cos(2*pi*f1*i)]; %reference signal
s11=sign(ec*sin(2*pi*f1*(i-1))); %convergence rate adaptation criterion
s12=sign(ecv*sin(2*pi*f1*(i-2)));    %''
s13=sign(ecvv*sin(2*pi*f1*(i-3)));   %''
if s11~=s12&s12~=s13&mu1>0.00031     %''
   mu1=1/alf*mu1;                    %''
elseif mu1<0.00031                   %''
   mu1=0.0003;                       %''
end                                  %''
w1=w1+mu1*r1*ec;                     %filter weight update
y1=w1'*r1;                           %computation compensation signal

%second frequency
r2=[sin(2*pi*f2*i);cos(2*pi*f2*i)];
s21=sign(ec*sin(2*pi*f2*(i-1)));
s22=sign(ecv*sin(2*pi*f2*(i-2)));
s23=sign(ecvv*sin(2*pi*f2*(i-3)));
if s21~=s22&s22~=s23&mu2>0.00031
   mu2=1/alf*mu2;
elseif mu2<0.00031
   mu2=0.0003;
end
w2=w2+mu2*r2*ec;
y2=w2'*r2;

ecvv=ecv;
ecv=ec;
ec=e-y1-y2;    %computation compensated error signal
i=i+1;
```

Prediction function

```
function correc=predictfunc(delay)  %input is time from present moment
                                    %till the point of time of which
                                    %the error must be predicted [s]
global f1 f2 mu1 mu2 ec ecv ecvv w1 w2 ww1 ww2 alf i f1t f2t
phase1=2*pi*f1t*delay;          %conversion delay from seconds to radians
phase2=2*pi*f2t*delay;          %''
r1d=[sin(2*pi*f1*i+phase1);cos(2*pi*f1*i+phase1)]; %adaptation
r2d=[sin(2*pi*f2*i+phase2);cos(2*pi*f2*i+phase2)]; %reference signals
y1d=w1'*r1d;                    %computation compensation signal 1
y2d=w2'*r2d;                    %computation compensation signal 2
correc=y1d+y2d;                 %computation total compensation signal
```

Appendix C: C functions

repetitive.h

```
#ifndef _REPETITIVE_H
#define _REPETITIVE_H
#define ANC_T_SCALE 7
#define ANC_T_TO_TFE (0.898 * (1<<ANC_T_SCALE))
#define ANC_P_SCALE 10

typedef struct {
        int u1;                                 // correction signal
        int u2;                                 // ''
        int r1a;                                // reference signal
        int r1b;                                // ''
        int r2a;                                // ''
        int r2b;                                // ''
        int e;                                  // error value
        int ec;                                 // corrected error value
        int ecv;                                // corrected error value 1 sample earlier
        int ecvv;                               // corrected error value 2 samples earlier
        int w1a, w1b, w2a, w2b, mu1, mu2; // filter weights and convergence rates
        int i,j;                                // indices for sin and cosine function
        int correc;                             // last correction computed by the predict function
        int d;                                  // delay for which last correction was computed
} t_anc_param;

//prototypes
int anc_init (t_anc_param*);
int anc_updatefunc (t_anc_param*, int);
int anc_predictfunc (t_anc_param*, int);

#endif
```

repetitive.c

```
#include <stdio.h>
#include <math.h>
#include "repetitive.h"

#define ANC_MAX_LOGS 10000    // number of entries in debug array
#define pi 3.14159265358979
#define alf 1.5               // factor of decrease of convergence rate
#define f1t 0.66              // first frequency in cycles/second
#define w1ts (int)(2*pi*f1t*1024)   // scaled version of f1t
#define f1 f1t/500.0          // first frequency in cycles/sample
#define w1s (int)(2*pi*f1*1024)     // scaled version of f1
#define f2t 1.31              // second frequency in cycles/second
#define w2ts (int)(2*pi*f2t*1024)   // scaled version of f2t
#define f2 f2t/500.0          // second frequency in cycles/sample
#define w2s (int)(2*pi*f2*1024)     // scaled version of f2

const int sinus[102] = {0, 64, 128, 192, 255, 316, 377, 436, 493, 549, 602, 653, 701, 746, 789, 828,
                        865, 897, 927, 952, 974, 992, 1006, 1016, 1022, 1024, 1022, 1016, 1006, 992,
                        974, 952, 927, 897, 865, 828, 789, 746, 701, 653, 602, 549, 493, 436, 377, 316,
                        255, 192, 128, 64, 0, -64, -128, -192, -255, -316, -377, -436, -493, -549, -602,
                        -653, -701, -746, -789, -828, -865, -897, -927, -952, -974, -992, -1006, -1016,
                        -1022, -1024, -1022, -1016, -1006, -992, -974, -952, -927, -897, -865, -828,
                        -789, -746, -701, -653, -602, -549, -493, -436, -377, -316, -255, -192, -128, -64, 0, 0};

static int sintab( int );             // sine function with the use of a table
static int costab( int);              // cosine function with the use of a table

// stuff for debug array
static t_anc_param data[ANC_MAX_LOGS];   // cyclic buffer
```

```
static int data_ind = 0;                    // points to next debug entry to fill
static int data_full = 0;                   // true if all entries have been filled
int data_log = 1;                           // true if data is being logged

void anc_add_to_log (t_anc_param *anc)
{
        data[data_ind] = *anc;
        data_ind++;
        if (data_ind == ANC_MAX_LOGS)
         {
           data_ind = 0;
           data_full = 1;
                data_log = 0;
         }
}

void anc_dump_data (void)
{
 int i;
 FILE    *fp = NULL;
 data_log = 0;                              // stop logging while saving data
 fp = fopen("logs/anc.dat", "w");
 if (fp != NULL )
 {

fprintf(fp,"%%u1\tu2\tr1a\tr1b\tr2a\tr2b\tec\tecv\tecvv\tw1a\tw1b\tw2a\tw2b\tmu1\tmu2\ti\tj\tcorrec\td\n");
   if (data_full)
    {
        for (i=data_ind; i < ANC_MAX_LOGS; i++)
        {
           fprintf (fp, "%i\t%i\t%i\t%i\t%i\t%i\t%i\t%i\t%i\t%i\t%i\t%i\t%i\t%i\t%i\t%i\t%i\t%i\t%i\t%i\t%i\n",
                                data[i].u1,
                                data[i].u2,
                                data[i].r1a,
                                data[i].r1b,
                                data[i].r2a,
                                data[i].r2b,
                                data[i].e,
                                data[i].ec,
                                data[i].ecv,
                                data[i].ecvv,
                                data[i].w1a,
                                data[i].w1b,
                                data[i].w2a,
                                data[i].w2b,
                                data[i].mu1,
                                data[i].mu2,
                                data[i].i,
                                data[i].j,
                                data[i].correc,
                                data[i].d);
         }
        }
  for (i=0; i<data_ind; i++)
   {
        fprintf (fp, "%i\t%i\t%i\t%i\t%i\t%i\t%i\t%i\t%i\t%i\t%i\t%i\t%i\t%i\t%i\t%i\t%i\t%i\t%i\t%i\t%i\n",
                        data[i].u1,
                        data[i].u2,
                        data[i].r1a,
                        data[i].r1b,
                        data[i].r2a,
                        data[i].r2b,
                        data[i].e,
                        data[i].ec,
                        data[i].ecv,
                        data[i].ecvv,
                        data[i].w1a,
```

```
                    data[i].w1b,
                    data[i].w2a,
                    data[i].w2b,
                    data[i].mu1,
                    data[i].mu2,
                    data[i].i,
                    data[i].j,
                    data[i].correc,
                    data[i].d);
    }
   fclose (fp);
  }
  data_full = 0;
  data_ind = 0;
  data_log = 1;                    // start logging again
}

// function anc_init ()
int anc_init (t_anc_param* anc_param)
{
        anc_param->mu1=(int)(0.02*1024*64);      // initial values
        anc_param->mu2=(int)(0.02*1024*64);
        anc_param->ec=1;
        anc_param->ecv=0;
        anc_param->ecvv=0;
        anc_param->w1a=anc_param->w1b=anc_param->w2a=anc_param->w2b=(int)(0.001*1024);
        anc_param->i=1;
        anc_param->j=1;
        return (1);
}


// function ec=updatefunc(e)
int anc_updatefunc (t_anc_param* anc_param, int e)
{
        int r1a, r1b,  s11, s12, s13, r2a, r2b, s21, s22, s23, u1,u2;
        int i,j;

        // save error for data analysis
        anc_param->e = e;

        // compute filter for first reference signal
        i = anc_param->i;

        r1a=sintab(w1s*i);
        r1b=costab(w1s*i);
        anc_param->r1a = r1a;
        anc_param->r1b = r1b;

        // convergence rate 1 adaptation criterion
        if (anc_param->ec*sintab(w1s*(i-1)) >0)
        {
                s11=1;
        }
        else
        {
                s11=-1;
        }

        if (anc_param->ecv*sintab(w1s*(i-2)) >0)
        {
                s12 =1;
        }
        else
        {
                s12 = -1;
        }
```

```
if (anc_param->ecvv*sintab(w1s*(i-3)) >0)
{
        s13 = 1;
}
else
{
        s13 = -1;
}

if (s11!=s12 && s12!=s13 && anc_param->mu1 > (int)(0.00031*1024*64))
{
 anc_param->mu1=(2*( anc_param->mu1)/3);
}
else
 if ( anc_param->mu1< (int)(0.00031*1024*64))
  {
   anc_param->mu1=(int)(0.0003*1024*64);
  }

// filter weight 1 adaptation
anc_param->w1a=anc_param->w1a+((anc_param->mu1*r1a)/64*(anc_param>ec/1024))/1024;
anc_param->w1b=anc_param->w1b+((anc_param->mu1*r1b)/64*(anc_param>ec/1024))/1024;
u1= anc_param->w1a*r1a/1024 + anc_param->w1b*r1b/1024;  // correction signal 1
anc_param->u1 = u1;

if (i==758)                    // index update 1
{
        anc_param->i=1;
}
else
{
        anc_param->i++;
}


// compute filter for second reference signal
j = anc_param->j;

r2a=(int)(sintab(w2s*j));
r2b=(int)(costab(w2s*j));
anc_param->r2a = r2a;
anc_param->r2b = r2b;


// convergence rate 2 adaptation criterion
if ( anc_param->ec*sintab(w2s*(j-1))>0)
{
        s21 =1;
}
else
{
        s21 = -1;
}

if  (anc_param->ecv*sintab(w2s*(j-2))>0)
{
        s22 = 1;
}
else
{
        s22 = -1;
}

if (anc_param->ecvv*sintab(w2s*(j-3))>0)
{
        s23 = 1;
```

```
        }
        else
        {
                s23 =-1;
        }

        if (s21!=s22 && s22!=s23 && anc_param->mu2>(int)(0.00031*1024*64))
        {
          anc_param->mu2=(int)(2*( anc_param->mu2)/3);     }
        else
          if ( anc_param->mu2<(int)(0.00031*1024*64))
            {
              anc_param->mu2=(int)(0.0003*1024*64);
            }

        // filter weight 2 adaptation
        anc_param->w2a=anc_param->w2a+((anc_param->mu2*r2a)/64*(anc_param>ec/1024))/1024;
        anc_param->w2b=anc_param->w2b+((anc_param>mu2*r2b)/64*(anc_param>ec/1024))/1024;
        u2= anc_param->w2a*r2a/1024 + anc_param->w2b*r2b/1024; //correction signal 2
        anc_param->u2 = u2;


        if (j==382)                     // index update 2
        {
                anc_param->j=1;
        }
        else
        {
                anc_param->j++;
        }

        // save errors for next time
        anc_param->ecvv= anc_param->ecv;
        anc_param->ecv= anc_param->ec;
        anc_param->ec=e*1024-u1-u2;       // corrected error


        // log some data for analysis
        if (data_log)
          anc_add_to_log (anc_param);

        return (r2a );
}

// function predictfunc
// d : delay in seconds * 128
int anc_predictfunc (t_anc_param* anc_param, int d)
{
        int i, j;
        int phase1, phase2, r1ad, r1bd, r2ad, r2bd, u1d, u2d;

        anc_param->d = d;
        i = anc_param->i;
        j = anc_param->j;

        phase1=((w1ts)/64*d) % (int)((2*pi)*(1<<(ANC_T_SCALE+4))); // phase1 calculation from delay
        phase2=((w2ts)/128*d) % (int)((2*pi)*(1<<(ANC_T_SCALE+3)));//phase2 calculation from delay

        r1ad=sintab(w1s*i+phase1);  // reference signal 1a with delay
        r1bd=costab(w1s*i+phase1); // reference signal 1b with delay
        u1d=anc_param->w1a*r1ad/1024+anc_param->w1b*r1bd/1024; //correction signal 1 with delay

        r2ad=sintab(w2s*j + phase2); // reference signal 2a with delay
        r2bd=costab(w2s*j + phase2);// reference signal 2b with delay
        u2d=anc_param->w2a*r2ad/1024+anc_param->w2b*r2bd/1024; //correction signal 2 with delay

        anc_param->correc=u1d+u2d;// total correction signal
```

```
        return (anc_param->correc);
}

// function sine
static int sintab (int phase)
{
        int position;

        position = phase /64;

        return (sinus[position]);
}

// function cosine
static int costab (int phase)
{
        int position;

        position = phase /64;
        if (position<=75)
        {
                position=position+25;
        }
        else
        {
                position=position-75;
        }

        return (sinus[position]);
}
```

References

[1] T. de Hoog, "Stability and performance of memory loop filtered control systems", Pato course, 1998

[2] M. Tomizuka, "Zero phase error tracking algorithm for digital control", Trans. of ASME Journal of dynamic systems, Measurement and control, Vol. 109, March 1987, pp 65-68

[3] S.M. Kuo, D.R. Morgan, "Active Noise Control Systems", Chichester: Wiley-Interscience, 1996

[4] R.W. Harris, D.M. Chabries, F.A. Bishop, "A variable step (VS) adaptive filter algorithm", IEEE Trans. acoust., speech, signal processing, ASSP-34, pp 309-316, April 1986

[5] O.G. Smink, "Practical application of repetitive control", Océ-Technologies B.V. R&D internal report, 2002

[6] B. Widrow, S.D. Stearns, "Adaptive signal processing", London: Prentice Hall, 1985

[7] H. Schildt, "C++ The Complete Reference", London: Osborne McGraw-Hill, 1995