Notes on Adaptive Feedforward

Gustafsson, Kjell

1987

Document Version:
Publisher's PDF, also known as Version of record

Link to publication

Citation for published version (APA):
Gustafsson, K. (1987). *Notes on Adaptive Feedforward*. (Technical Reports TFRT-7346). Department of Automatic Control, Lund Institute of Technology (LTH).

Total number of authors:
1

CODEN: LUTFD2/(TFRT-7346)/1-14/(1987)

Notes on Adaptive Feedforward

Kjell Gustafsson

Department of Automatic Control
Lund Institute of Technology
March 1987

Department of Automatic Control
Lund Institute of Technology
P.O. Box 118
S-221 00 Lund Sweden

Document name
Report

Date of issue
March 1987

Document Number
CODEN:LUTFD2/(TFRT-7346)/(1-14/(1987)

Author(s)
Kjell Gustafsson

Supervisor
Karl Johan Åström

Sponsoring organisation

Title and subtitle
Notes on Adaptive Feedforward

Abstract

This report treats the simultaneous use of adaptive feedforward and a self-tuning regulator. Two different methods to determine the feedforward are derived, one direct and one indirect. The indirect method shows promising results for low order plants, while there were problems making the direct method function properly.

Key words
adaptive, feedforward, self-tuning regulator

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title

ISBN

Language
English

Number of pages
14

Recipient's notes

Security classification

1. Introduction

Disturbances are often a problem when trying to control a plant. Sometimes the disturbance, or a signal strongly correlated with it, can be measured. It can then be used for feedforward, which cancels the disturbance. Although theoretically feasible this may be hard to do, especially if the plant dynamics are unknown and/or varying. One often rests content with constant feedforward. In this report we will investigate the possibility to simultaneously apply adaptive feedforward and adaptive feedback to an unknown plant.

Problem setup

Assume having a plant that can be modeled as shown in Figure 1. There is an input u, an output y and a disturbance v. The disturbance is measurable. $G_1(p)$ and $G_2(p)$ are unknown rational functions in the differential operator p. A self-tuning regulator (STR) (Åström, 1987) will be used to control the plant, while the effects of the disturbance will be reduced by adding a feedforward. The result will be a system as in Figure 2.

The two parts will be dealt with separately. First a controller is designed as if the disturbance was not present and then the feedforward is used to remove the effects of the disturbance. The controller and the feedforward will be implemented as discrete time systems. Their outputs will be fed to the plant through a zero order hold circuit.

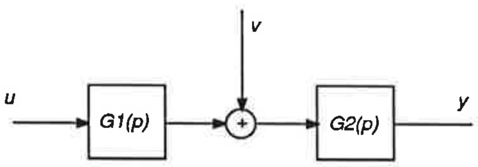

Figure 1. Plant model

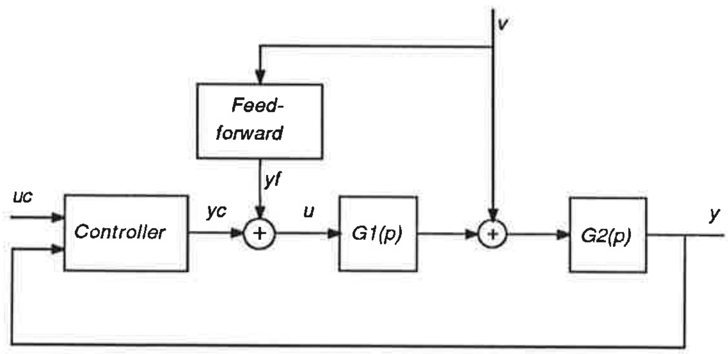

Figure 2. Plant with controller and feedforward.

2. The controller

The actions of the STR can be divided into two parts. An estimate of the plant is formed, and then a controller is designed as if the estimate was the true plant.

Estimation

The plant is estimated with the method of recursive least squares (RLS) (Ljung, 1983), using sampled versions of the signals and the plant model

$$y = \frac{B(q)}{A(q)} y_c$$

where A and B are polynomials in the forward shift operator q, which satisfy the degree relation, $\deg A = \deg B + 1$.

The disturbance v is neglected when estimating the plant. The feedforward will make its effect on y small, and therefore v is disregarded. If the feedforward does not succeed to fully cancel the disturbance there will be a term in y caused by v. This gives a bias in the estimate of A and B. Hopefully the bias will be sufficiently small not to pose any problems in the controller design.

The plant is estimated in closed loop, i.e. y_c depends on y. One might fear ending up estimating the inverse of the controller instead of the true plant. It can however be shown (Söderström, 1984) that the true plant will indeed be estimated provided that the controller has sufficiently high order or the set point is time varying.

Another possibility would be to include v in the estimation and estimate a plant on the form $\tilde{A}(q)y = \tilde{B}(q)y_c + \tilde{C}(q)v$. This adds complexity to the estimator and it is unclear if it gives any benefits.

Controller design

Since a discrete-time model of the plant is estimated, it is natural to use a discrete time controller. A controller on the form

$$R(q)y_c = T(q)u_c - S(q)y$$

is used, where R, S and T are polynomials in the forward shift operator q, and u_c is the reference signal. To get a realizable controller we must have $\deg S \leq \deg R$ and $\deg T \leq \deg R$. Neglecting the disturbance v, the closed loop system becomes

$$y = \frac{BT}{AR + BS} u_c$$

Choosing neither to cancel any zeros of the plant, nor to introduce any new ones, gives the following equations to be satisfied.

$$\begin{aligned} AR + BS &= A_m A_o \\ T &= b' A_o \end{aligned}$$

A_m is a monic polynomial containing the desired closed loop poles and A_o is the observer polynomial. The constant b' is chosen to get static gain from u_c to y equal to one.

These equations are solvable given A, B, A_m and A_o. The estimator gives A and B, and A_m and A_o will be chosen corresponding to continuous-time Butterworth polynomials of lowest possible degree.

Sampling rate

The sampling rate (ω_s) has to be decided. If chosen to low, long time intervals between control actions will lead to poor performance. If on the other hand chosen to large, there will be little development between successive samples. The estimator will give a bad estimate, which also will lead to bad performance. Quite heuristically we will take $\omega_s \approx 20\omega_c$ ($h\omega_c \approx 0.3$), where ω_c is the bandwidth of the closed loop system (Åström, 1985).

3. The feedforward

To design the feedforward the dependence between y and v has to be known. Since the plant is unknown this dependence has to be estimated. Exactly how the estimation is done will be dealt with later. The estimation yields a discrete-time transfer function $H_2(q)$ describing the affect of v on y.

When estimating discrete-time models of continuous-time systems the result is a function of both the continuous-time system and the excitation signal. In the controller the transfer function between u and y is estimated as $B(q)/A(q) = H_1(q)$. Ideally it will be a sampled version of G_1G_2. Both H_1 and H_2 contain a description of G_2, but due to different excitation signals these descriptions will in general not be equal. How similar the "common" parts (corresponding to G_2) will be depends on G_1 and the frequency contents of the signals u and v. The estimation of the plant as H_1 and H_2 will thus not preserve the structure in Figure 1, but instead lead to an estimated system as in Figure 3.

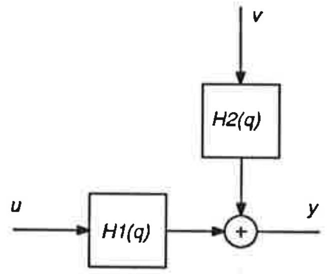

Figure 3. Sampled plant model

By introducing the controller and the feedforward we get a complete system as in Figure 4. If H_1 and H_2 were known it would be obvious from the figure how to choose the feedforward $H_f(q)$. Setting $H_2(q) = D(q)/C(q)$, where $C(q)$ and $D(q)$ are polynomials in the forward shift operator q (deg$D \leq$ deg$C$), yields

$$H_f(q) = -\frac{H_2(q)}{H_1(q)} = -\frac{AD}{BC}$$

A straightforward method to determine H_f would be to use the controller's estimate of H_1 together with an estimate of H_2 and then calculate H_f according to the formula above. Another possibility is to construct signals such that H_f can be estimated directly. In the sequel the former will be refered to as indirect estimation and the latter as direct estimation.

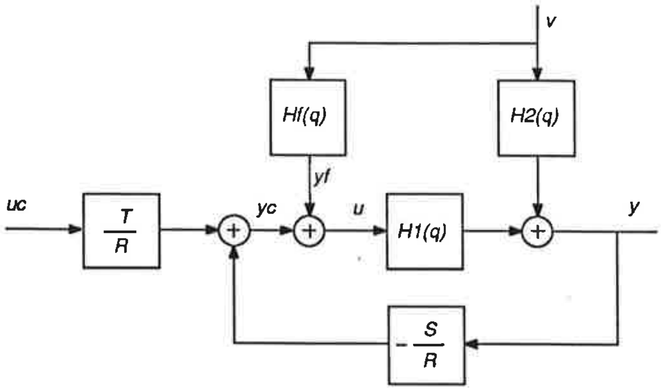

Figure 4. Sampled plant model with controller and feedforward.

Indirect estimation of the feedforward

In this method H_f is calculated from the estimates of H_1 and H_2. The estimation of H_1 is already done in the controller and here only H_2 has to be dealt with. As with H_1 RLS will be used as estimation method.

What signals should be used to estimate H_2? If we try to use v and y we will fail to get a good estimate. The transfer function H_2 is not the only path from v to y. Another one is introduce through u via H_1, since u depends on v both through the feedback and the feedforward. The remedy is to try to construct the signal that would be the output if v was fed through the true H_2. It can be done by feeding u through an estimate of H_1 and then subtract it from y. Now v and the calculated signal can be used to estimate H_2. The only limitation is that H_1 has to be stable. Figure 5 shows how the estimators are connected to the plant.

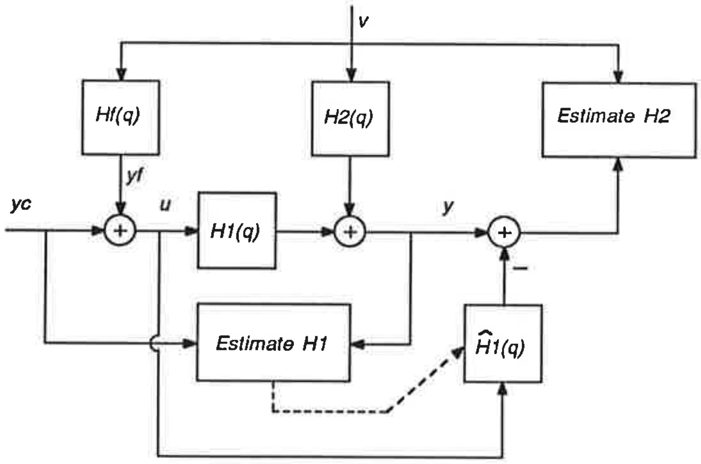

Figure 5. Connection of the estimators to the plant (indirect estimation).

The order of the system is not known. A model of too high order gives common modes in H_1 and H_2. These must be canceled before trying to realize H_f. Almost common modes between H_1 and H_2 may arise since both H_1 and H_2 contain descriptions of G_2. Such modes also have to be canceled before realizing H_f.

Some precautions has to be taken when the system is started up. The estimation of H_1 is started directly since it is needed to calculate the controller.

When the estimate of H_1 is assumed to have converged the estimation of H_2 is started, and when that estimation also begins to converge the feedforward is connected to the plant. During startup the feedforward is set to zero.

Direct estimation of the feedforward

If the disturbance enters early in the plant, i.e. G_1 is of low order compared to G_2, then H_1 and H_2 may have modes that are similar. It can be hard, especially if H_1 and H_2 are of high order, to find and cancel such modes when calculating H_f. It would therefore be nice to be able to estimate H_f directly instead of calculating it. One could then start estimating a feedforward of order zero and then step by step increase its order until a sufficiently good feedforward was achieved.

The block diagram in Figure 4 is manipulated in order to see how to estimate H_f directly. Take H_1 and move it leftwards out in the two branches left of the summation point (Figure 6). Change the order of H_f and H_1. Formally this can not be done since at least H_f is time varying, but when H_f has converged the variation is so slow that it is safe to change the order. After some further manipulations the system shown in Figure 7 is derived. In the figure a block with the transfer function H_2/H_1 $(-H_f)$ can be seen. If the signals entering and leaving this block are constructed, H_f can be estimated directly.

By using an estimate of H_1 (constructed in the controller) and the knowledge of the currently used feedforward, the system in Figure 8 is constructed. The estimator is implemented using RLS. To be able to use this method H_1 must be stable.

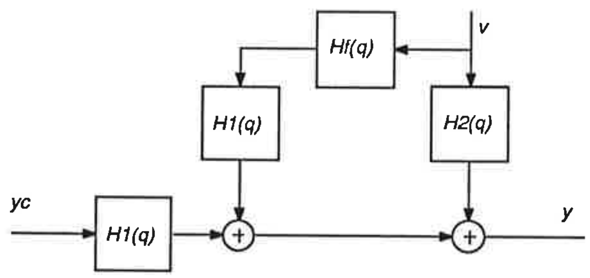

Figure 6. H_1 moved leftwards in the original block diagram.

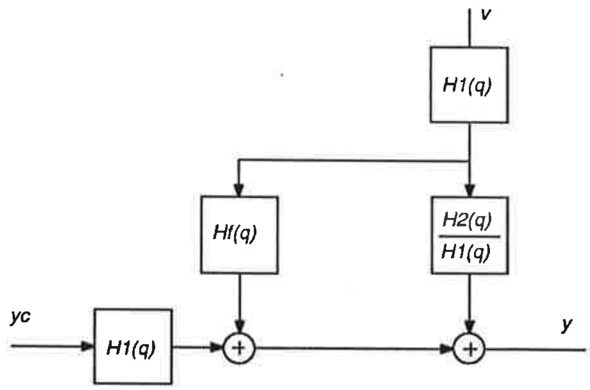

Figure 7. Final block diagram

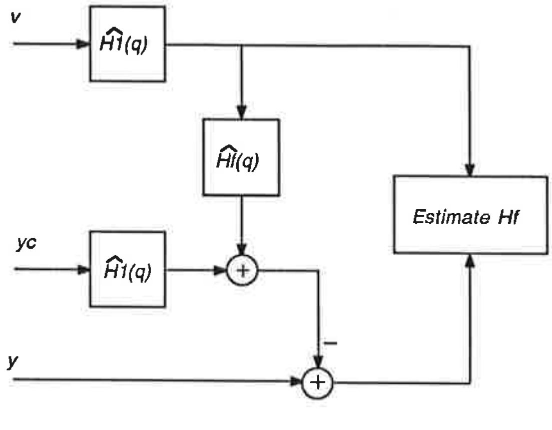

Figure 8. Direct estimation of H_f.

The method has a property that may lead to trouble. The signal v is filtered through an estimate of H_1 before it is fed to the estimator. Since H_1 probably has low pass character, high frequency components in v will be removed. It will yield a bad estimate of the high frequency parts of H_f. Such uncertainty in H_f will reduce the ability to cancel disturbances with high frequencies.

In this method as well as in the former some precautions has to be taken when starting up the system. The estimation of H_1 is started directly, and when it starts to converge the estimation of H_f is begun. When the estimation of H_f begins to converge the feedforward is connected to the plant. During startup the feedforward is set to zero.

Limitations

How much will be achieved with H_f as above? The question is hard to answer since the success of the feedforward depends on several things. By introducing a feedforward with dynamics we hope to be able to cancel more than just constant disturbances. We will now discuss a couple of possible pitfalls.

H_f is discrete-time and it is recognized that there is an upper bound on the frequency components in v that may be canceled. A naive first thought is to regard half the sampling rate ($\omega_s/2$) as this bound. The bound is unfortunately much more restrictive than that. When estimating H_1 and H_2 as discrete ARMA-models the result will depend on the excitation signals. Since the control signal u is piece-wise constant, H_1 will be a sampled version of G_1G_2. It is only valid for this type of input signals. When constructing the feedforward the signal v is fed through H_2/H_1, i.e. $1/H_1$ is used as an approximation of the inverse of the plant. This inverse is only valid for signals that are piece-wise constant and will therefore be a bad approximation if v varies substantially during the sampling interval. Thus to get successful feedforward v should not contain high frequency components. Its hard to specify an exact bound on the frequency contents in v, but it will certainly be well below $\omega_s/2$.

The restriction of v to just contain low frequency components (slowly varying or constant) will make the need for dynamics in H_f less pronounced. It may be adequate, and easier, to just adapt a constant feedforward ($H_f = K$) and/or put an integrator in the controller to handle load disturbances.

Another problem is that H_f may be nonproper or contain unstable poles.

It will then be unrealizable. The unstable poles can be handled by substituting them with there reflection in the unit circle. It will leave the amplitude characteristics of H_f unchanged but alter the phase characteristics. The procedure is actually optimal for step disturbances (Åström, 1976). By regarding the unproperness as poles at infinity it may be handled as the unstable poles. Reflecting poles at infinity in the unit circle yields poles at the origin. Unproperness is thus handled by adding poles at the origin until H_f becomes proper. This is equal to adding delay to H_f.

4. Simulations

The two methods described above were simulated using SIMNON. A Pascal system was added to handle all polynomial manipulations (see appendix).

Among others the following three setups were simulated:

$$
\begin{aligned}
&\text{Setup 1:} \quad G_1(s) = 1, && G_2(s) = \frac{0.1}{s+0.1} \\
&\text{Setup 2:} \quad G_1(s) = 1, && G_2(s) = \frac{0.06}{s^2+0.12s+0.06} \\
&\text{Setup 3:} \quad G_1(s) = \frac{0.1}{s+0.1}, && G_2(s) = \frac{0.1}{s+0.1}
\end{aligned}
$$

All discrete-time parts of the system had the same sampling period (h). It was chosen to 1.

The controller estimates H_1 as a second order system. It tries to cancel common modes and depending on the resulting degree of H_1 it choses A_m and A_o as

$$
\begin{aligned}
\deg H_1 = 1 \Rightarrow A_m &= q - 0.8187 && (\Leftrightarrow h/T = 0.2) \\
A_o &= 1 \\
\deg H_1 = 2 \Rightarrow A_m &= q^2 - 1.7189q + 0.7536 && (\Leftrightarrow \zeta = 0.7, \quad \omega h = 0.2) \\
A_o &= q - 0.6065 && (\Leftrightarrow h/T = 0.5)
\end{aligned}
$$

The chosen A_m and A_o corresponds to continuous-time polynomials on the following form: $s + h/T$, $s^2 + 2\zeta\omega h s + (\omega h)^2$.

The disturbance signal is created by feeding white discrete-time noise through a second order filter.

$$
\begin{aligned}
&v(t) = \frac{10(0.2h)^2}{(s+0.2h)^2} e(t) \\
&e(t) = e'(k), \quad k \le t < k+h, \quad e'(k) \in \mathrm{N}(0, 0.1)
\end{aligned}
$$

To be able to handle a time-varying plant the estimators have to forget old data. A forgetting factor (λ) of 0.998 was used in the RLS-algorithms.

In the indirect method H_2 is estimated as a second order system. Together with H_1 (also of second order) this gives a H_f of order four. Often there will be pole-zero cancellations in H_f, giving a feedforward of lower order. When using the direct method one can chose to estimate a H_f of order zero, one or two.

The controller and the estimation of H_1 was started at $t = 0$. At $t = 100$ the estimation of the feedforward was started and at $t = 300$ the feedforward link was connected to the rest of the system.

Results for the indirect method

This method performs quite well. As one could expect it takes some time before the estimates have converged and an adequate controller and feedforward are found. A simulation of setup 3 is shown in Figure 9. There is an initial transient but soon the controller makes the plant output track the desired signal. Due to the disturbance the tracking is not prefect ($100 \leq t \leq 300$). At $t = 300$ the feedforward is connected to the plant. Almost immediately the disturbance is completely canceled, although this requires forming a feedforward that is very close to a dierentiation (G $_1$ low pass). Note how the control signal u changes character when the feedforward is connected.

When trying the method on plants of higher orders it does not perform as well as in Figure 9. It is hard to handle the cancellation of common factors in H_f in a good way. Often there is poles and zeros that are very close. When the estimates vary (due to noise or dierent excitation signals) the poles and zeros will move about. Depending on how close they are they will sometimes be canceled and sometimes not. When changing the order of the feedforward one do not instantaneously get exactly the right parameters in H_f. This is often enough excitation to make the poles and zeros move, making H_f change order again.

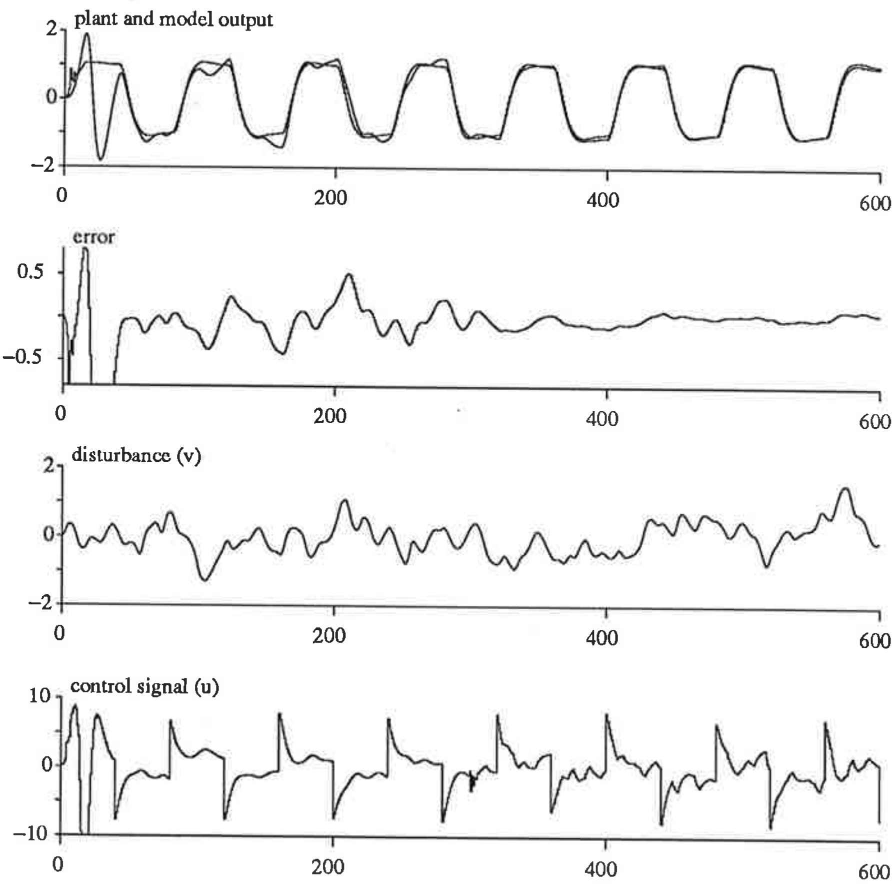

Figure 9. Successful use of the indirect method (setup 3). The feedforward is connected at $t = 300$.

Results for the direct method

With this method it was hoped to solve the problems encountered in the indirect method. Setup 1 and 2 works quite well but as soon as G_1 contain some dynamics the method fails to find the adequate H_f. The feedforward converges to something different than expected and often it makes the system performance worse instead of making it better. A typical simulation (setup 3) is shown in figure 10. It is easy to see how the performance degrades when the feedforward is connected ($t = 300$).

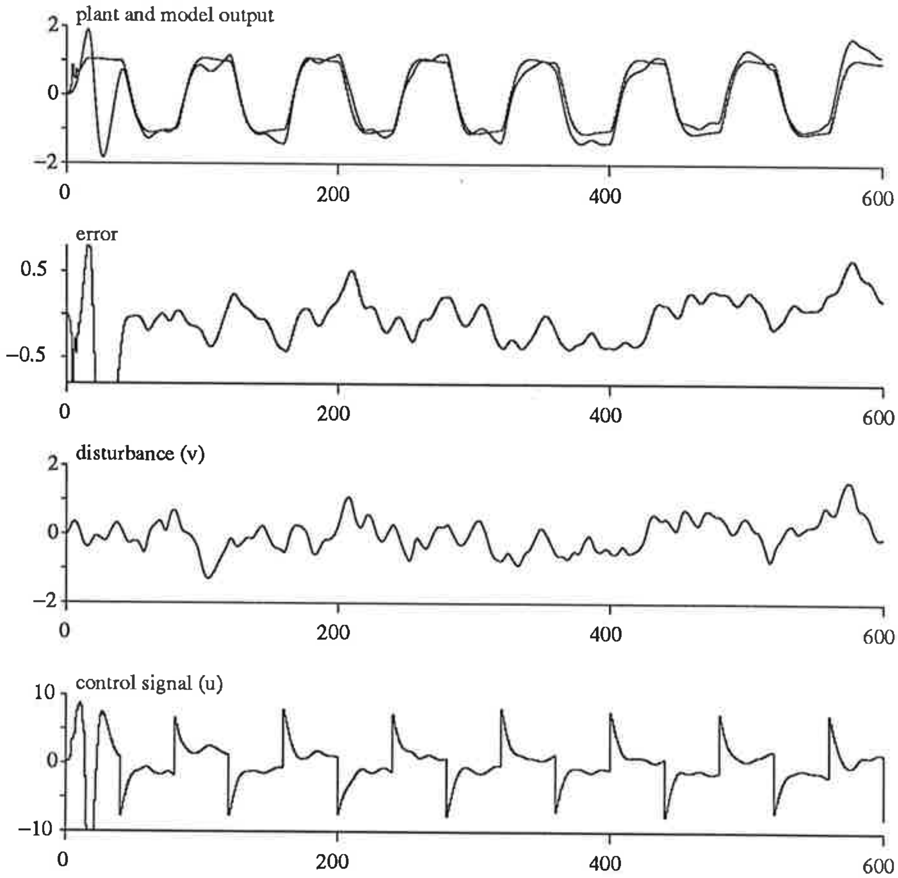

Figure 10. Unsuccessful use of the direct method (setup 3). The feedforward is connected at $t = 300$.

5. Conclusions

The indirect method performs quite well for plants of low order. When the plant order increases pole-zero cancellations give rise to problems. A possible way of avoiding such problems would be to estimate the feedforward directly. The direct method derived here was unfortunately not successful, although the same method have been used in other types of problems and has been reported to perform well (Widrow, 1985). The reason for the malfunction is not understood.

Further work

The indirect method shows that it is possible to get quite successful adaptive feedforward. The problems with pole-zero cancellations suggests the use of a direct method for higher order plants. One reason to the problems encountered with the direct method may be the way the signals used to estimate H_f are synthesized. The signal v is filtered through H_1 even though it is not a piece-wise constant signal. If v vary substantially during the sampling interval then H_1 will not be a good process model for this signal type. It would be interesting trying to attack the problem by estimating continuous time parameters, i.e. estimate $G_1(p)G_2(p)$ and $G_2(p)$, instead of parameters in ARMA-models. When having these estimates one can either calculate a continuous time controller and a continuous time feedforward or sample the model and use discrete time synthesis. It would even be possible to choose different sampling rates in the controller and the feedforward.

6. References

ÅSTRÖM, K. J. (1976): *Reglerteori*, AWE/GEBERS, Stockholm, pp. 242–244.

ÅSTRÖM, K. J. AND B. WITTENMARK (1984): *Computer Controlled Systems*, Prentice-Hall, Englewood Cliffs, p. 178.

ÅSTRÖM, K. J. AND B. WITTENMARK (To appear): *Adaptive Control*, Prentice-Hall, Englewood Cliffs.

LJUNG, L. AND T. SÖDERSTRÖM (1983): *Theory and Practice of Recursive Identification*, The MIT Press, Cambridge, Massachusetts.

SÖDERSTRÖM, T. (1984): *Lecture notes in identification*, Uppsala universitet, pp. 8:13–8:14.

WIDROW, B. AND SAMUEL D. STEARNS (1985): *Adaptive Signal Processing*, Prentice-Hall, Englewood Cliffs, pp. 288–294.

Appendix

When designing the controller and the feedforward there is a need to do a lot of polynomial manipulations. To handle these manipulations a Pascal system was included in Simnon. The polynomial routines used were in large "borrowed" or inspired from PCALC.

The routines can be divided into three groups. The first group contains basic routines as polynomial arithmetic, different tests, evaluation etc. The second group uses the routines in the first group to calculate greatest common divisors, least common multiples and to do spectral factorization. In the third group we only find two routines; one to do the controller design and one to do the feedforward design. The feedforward design routine appears in two different versions. One to be used in the direct method and another to be used in the indirect.

First group:

```
function NullPoly(poly : PolynomialType;
                  eps  : CoefficientType) : boolean;
  { Tests if a polynomial 'poly' is null. }

function RelativeDegree(poly1,poly2 : PolynomialType) : integer;
  { Calculates the relative degree between 'poly1' and
    'poly2'. }

function PolyNorm(poly : PolynomialType) : CoefficientType;
  { Calculates the absolute value of the biggest coefficient
    in a polynomial. }

procedure Reduce(var poly : PolynomialType);
  { Reduces the order of a polynomial by discarding terms that
    are too small. If the degree of a polynomial happens to be
    negative it's set to zero. }

function PolyEval(poly  : PolynomialType;
                  arg   : ArgumentType) : ArgumentType;
  { Evaluates the polynomial 'poly' with the argument 'arg'. }

procedure PolyDef(var poly   : PolynomialType;
                      degree : DegreeType;
                      coeff  : CoefficientType);
  { Creates a polynomial on the form 'coeff*z^degree'. }

procedure PolySwap(var poly1,poly2  : PolynomialType);
  { Swaps 'poly1' and 'poly2'. }

procedure PolyOrder(var poly1,poly2  : PolynomialType;
                    var shift        : boolean);
  { Compares 'poly1' and 'poly2' and puts the one with the
    highest degree in 'poly1'. If the polynomials have the
    same degree, the one with the largest (absolute value)
    leading coefficient is put in 'poly1'. Wether a shift
```

```
    was done or not is marked in 'shift'. }

procedure Reciproc(     poly  : PolynomialType;
                    var reci  : PolynomialType);
  { Takes a polynomial and calculates its reciprocal. }


procedure PolyAdd(     term1,term2  : PolynomialType;
                   var sum          : PolynomialType);
  { Performes a polynomial addition. 'term1' and 'term2' are
    added and the result is placed in 'sum' (sum := term1 +
    term2). }

procedure PolySub(     term1,term2  : PolynomialType;
                   var difference   : PolynomialType);
  { Performes a polynomial subtraction. 'term2' is subtracted
    from 'term1' and the result is placed in 'difference'
    (difference := term1 - term2). }

procedure PolyMul(     factor1,factor2  : PolynomialType;
                   var product          : PolynomialType);
  { Performes a polynomial multiplication. 'term1' and 'term2'
    are multiplied and the result is placed in 'product'
    (product := factor1*factor2). }

procedure PolyDivMod(     numerator     : PolynomialType;
                          denominator   : PolynomialType;
                      var quotient      : PolynomialType;
                          remainder     : PolynomialType);
  { Performes a polynomial division. 'numerator' is divided
    by 'denominator' yielding 'quotient' and 'remainder'
    (numerator = quotient*denominator + remainder). }

procedure PolyDiv(     numerator     : PolynomialType;
                       denominator   : PolynomialType;
                   var quotient      : PolynomialType);
  {  By using the procedure 'PolyDivMod' this procedure
     performs a polynomial division. It should be used
     when one only wants to know the quotient. }

procedure PolyMod(     numerator     : PolynomialType;
                       denominator   : PolynomialType;
                   var remainder     : PolynomialType);
  { By using the procedure 'PolyDivMod' this procedure
    performs a polynomial division. It should be used
    when one only wants to know the remainder. }
```

Second group:

```
procedure gcd(    poly1,poly2 : PolynomialType;
              var common      : PolynomialType;
                  CommonEps   : CoefficientType);
  { Calculates the greatest common divisor of 'poly1' and
    'poly2'. The result is placed in 'common'. }

procedure gcdlcm(    poly1,poly2      : PolynomialType;
                 var p,q,r,s,common : PolynomialType;
                     CommonEps      : CoefficientType);
  { Calculates the greatest common divisor (gcd) and the
    least common multiple (lcm) of two polynomials 'poly1'
    and 'poly2'. While doing so a transformation matrix
    consisting of p, q, r and s is formed (row 1 = p r,
    row 2 = r s). The matrix describes how to get the gcd
    and the lcm. The gcd is placed in 'common'.

    [poly1 poly2]*[p r] = [common 0]
                  [q s]

    lcm = poly1*r = -poly2*s
  }

procedure SpectralFactorize(    s         : PolynomialType;
                            var c         : PolynomialType;
                                mindiff   : CoefficientType);
  {             +    -                          +     -*
    Takes S = B (z)*B (z) and produces C = B (z)*B  (z).
    This is done with the same algorithm as in PCALC. }
```

Third group:

```
procedure RSTDesign(    apoly,bpoly         : PolynomialType;
                    var rpoly,spoly,tpoly   : PolynomialType;
                        CommonEps           : CoefficientType);
  { Takes the plant 'bpoly/apoly' and calculates a controller
    on polynomial form ('rpoly', 'spoly' and 'tpoly').
    'CommonEps' controls when to cancel common modes in the
    plant. }

procedure FeedForwardDesign(    apoly,bpoly  : PolynomialType;
                                cpoly,dpoly  : PolynomialType;
                            var qpoly,ppoly  : PolynomialType;
                                CommonEps    : CoefficientType;
                                SpecEps      : CoefficientType;
                                startfeed    : boolean);
  { Routine used in the indirect method. Calculates a
    feedforward  'qpoly/ppoly' from 'apoly', 'bpoly',
    'cpoly' and 'dpoly'. 'CommonEps' controls cancellation
    of common modes, and 'SpecEps' is used when reflecting
```

```
    unstable poles in the unit circle. 'startfeed' determines
    wether to connect the feedforward or not. }

procedure FeedForwardDesign(    qprimpoly      : PolynomialType;
                                pprimpoly      : PolynomialType;
                            var qpoly,ppoly    : PolynomialType;
                                CommonEps      : CoefficientType;
                                SpecEps        : CoefficientType;
                                startfeed      : boolean);
  { Routine used in the direct method. Calculates a
    feedforward 'qpoly/ppoly' from an estimated feedforward
    'qprimpoly/pprimpoly'. 'CommonEps' controls cancellation
    of common modes, and 'SpecEps' is used when reflecting
    unstable poles in the unit circle. 'startfeed' determines
    wether to connect the feedforward or not. }
```