SOME RESULTS CONCERNING FAST LINEAR MIMO ADAPTIVE INVERSE CONTROL

by

Gregory L. Plett

Technical Report EAS_ECE_2000_11

Department of Electrical and Computer Engineering

University of Colorado at Colorado Springs

Colorado Springs, CO 80933–7150

24 July 2000

Some Results Concerning Fast Linear MIMO Adaptive Inverse Control

Gregory L. Plett

Abstract

Adaptive inverse control is an automatic control-system design method which “learns” over time how to control a dynamic system (“plant”). The technique may be applied to single-input single-output (SISO) and multi-input multi-output (MIMO) plants. The plant may be linear or nonlinear. This report addresses preliminary results concerning a new way to perform adaptive inverse control of linear MIMO systems which learns very quickly.

I. Introduction

ADAPTIVE inverse control is an automatic control-system design method which learns how to control a specific plant [1–9]. Adaptive filtering methods are used throughout. Invisible to the user, a three-part process is used internally. First, an adaptive plant model $\widehat{P}$ learns the dynamics of the plant. Secondly, an adaptive feedforward controller C learns to control the dynamics of the plant. Thirdly, an adaptive feedback disturbance canceler X learns to cancel disturbances which affect the plant. These processes may proceed concurrently. A block diagram of an adaptive inverse control system is shown in Fig. 1.

Methods are known to quickly adapt controllers for linear SISO plants [6], but known methods for adapting controllers for MIMO plants learn very slowly [2, 3, 6]. This report presents preliminary results regarding a training method for linear MIMO adaptive inverse control that learns about as quickly as methods for linear SISO control.

II. Adaptive Digital Filtering

An adaptive filter is illustrated in Figure 2. It has an input, an output, and a “special input” called the desired response. The desired response d_k specifies the output we wish the filter to have. It is used to calculate an error signal e_k, which in turn is used to modify the internal parameters of the filter in such a way that the filter “learns” to perform a certain function. Often, the trick to applying adaptive filtering to a specific application is finding a way to generate an appropriate d_k signal.

For the sake of the present discussion we limit ourselves to *finite impulse response* (FIR) filters. The filter output is computed as a weighted sum of its current and N previous inputs

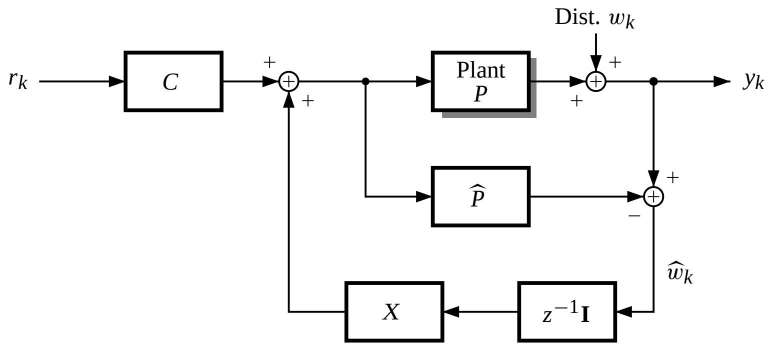


Fig. 1. Adaptive inverse control system block diagram.

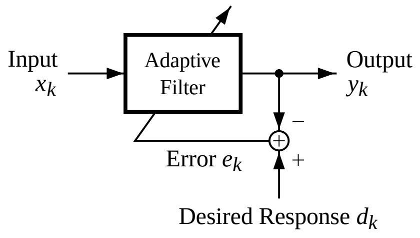


Fig. 2. Symbolic representation of an adaptive filter.

$$y_k = \mathbf{W}X_k, \tag{1}$$

where the column-vector

$$X_k = [x_k^T,\ x_{k-1}^T \ldots x_{k-N}^T]^T,$$

and **W** is the *weight matrix* of the filter.

Linear SISO filters have a single input and a single output at each time instant; for (1) to represent a SISO system, x_k and y_k must be scalars. **W** is then a single row, and its components are the impulse response of the filter. Linear MIMO filters have (possibly) many inputs and outputs at each time instant; this is also accommodated by (1) if x_k and y_k are (column) vector signals. **W** then has many rows, and the individual impulse responses are interleaved in each row.

The weights in **W** may be adapted in a variety of ways in order that the output of the filter learns to closely match the desired output. Adaptation is performed with the aid of the filter's desired response input signal d_k. At each time instant the filter output y_k is compared to this desired response, and the error is computed to be $e_k = d_k - y_k$. As the system runs, we wish to modify the weights of the filter in order to minimize the expected squared error. For example, the weights may be updated by a gradient-descent optimization procedure such as

$$\Delta\mathbf{W} = 2\mu e_k X_k^T,$$

where μ is a small positive *learning constant.* This rule, commonly known as the matrix-LMS algorithm, is well described in several textbooks [10, 11]. Other, more sophisticated, algorithms, such as matrix-RLS (described in Section IV. E on page 7) usually converge more quickly, and are very popular [12].

A nice property of linear filtering is that the optimal solution to which an adaptive process will converge is mathematically tractable if certain statistical information about the input and desired response is available. This solution is known as the *Wiener solution*. The SISO version is fully developed in [6], and the MIMO version is derived in Appendix A on page 12.

III. Linear SISO Adaptive Inverse Control

We may now consider adapting the feedforward controller C.[1] The goal is to make the dynamics of the controlled system PC approximate the fixed filter M as closely as possible, where M is a user-specified *reference model*. The input reference signal r_k is filtered through M to create a desired response d_k for the plant output. The measured plant output is compared with the desired plant output to create a system error signal $e_k^{(sys)} = d_k - y_k$. We wish to adapt C to minimize the mean-squared system error.

The reference model M may be designed in a number of ways. Following traditions of control-theory, we might design M to have a certain step response resembling a second-order system which meets design

[1] We shall restrict our development to apply only to stable plants. If the plant of interest is unstable, conventional feedback should be applied to stabilize it. Then the combination of the plant and its feedback stabilizer can be regarded as an equivalent stable plant.

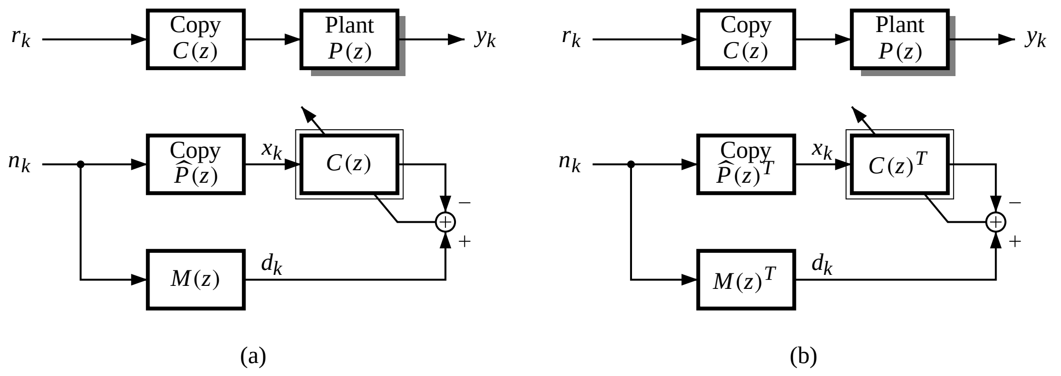


Fig. 3. Adapting a linear controller $C(z)$. (a) For a SISO linear plant; (b) For a MIMO linear plant. The plant modeling process has been omitted for clarity.

specifications. However, we can often achieve even better tracking control if we let M be simply a delay corresponding to the transport delay of the plant. The controller C will adapt to a delayed inverse of the plant dynamics.

We note that if the plant is minimum-phase, that is, has all of its poles and zeros inside the unit circle in the z-plane, then the inverse will be stable with all of its poles inside the unit circle. If the plant is nonminimum-phase, then some of the poles of the inverse will be outside the unit circle. According to the theory of two-sided z-transforms, the inverse will then either be unstable *or* noncausal. Since minimizing system error will not lead to an unstable solution—which would have unbounded system error—the algorithm will attempt to match a noncausal solution. This will result in very poor control unless the reference model has built-in latency. The longer the latency, the better we can approximate a delayed version of a noncausal inverse with a causal controller. A typical design latency is the transport delay of the plant.

A. Linear SISO Feedforward Control:

The challenge when attempting to adapt C is in generating its desired response signal. We notice that the system desired response is available at the output of the plant, and not at the output of the adaptive controller. A variety of solutions are known when the plant is a linear SISO system [6]. One simple and very effective method takes advantage of the commutability of transfer functions of linear SISO systems. That is, $P(z)C(z) = C(z)P(z)$. The block diagram of Figure 3(a) may then be used to adapt the controller. A random signal n_k is filtered by a digital copy[2] of the plant model $\widehat{P}(z)$ and then by the controller $C(z)$. It is also filtered by the reference model $M(z)$. Since the output of the reference model is the desired output of the cascade $C(z)\widehat{P}(z)$, it is used as the desired response signal for $C(z)$. A filter whose weights are a digital copy of $C(z)$ is used as the feedforward controller.

This method is unbiased by zero-mean disturbance, but may be affected by plant modeling errors. More sophisticated methods are available to adapt controllers and which are not affected by plant modeling errors [6]. If the RLS algorithm is used to adapt $C(z)$, convergence occurs within twice as many iterations as there are taps in the controller filter C [10].

[2] The weights of the digital copy are identical to the weights of the adaptive plant model, although the input to both filters is different and hence the outputs of the two filters are different.

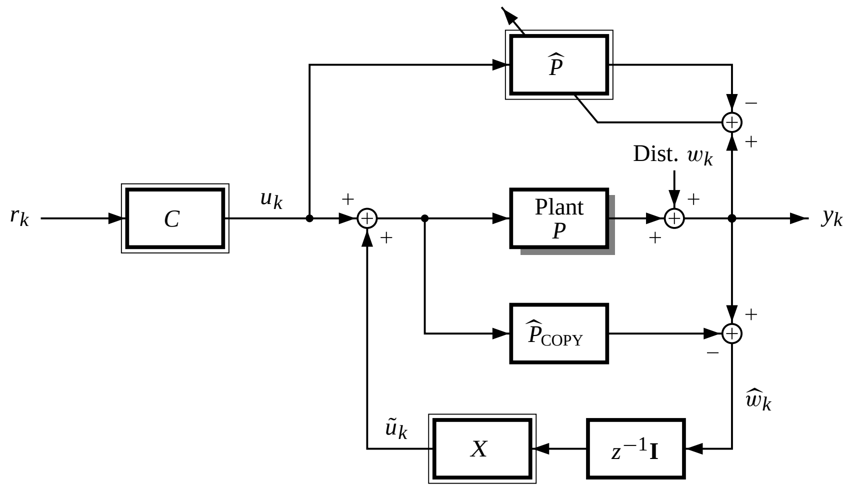


Fig. 4. Correct on-line adaptive plant modeling in conjunction with disturbance canceling for linear plants. The circuitry for adapting C has been omitted for clarity.

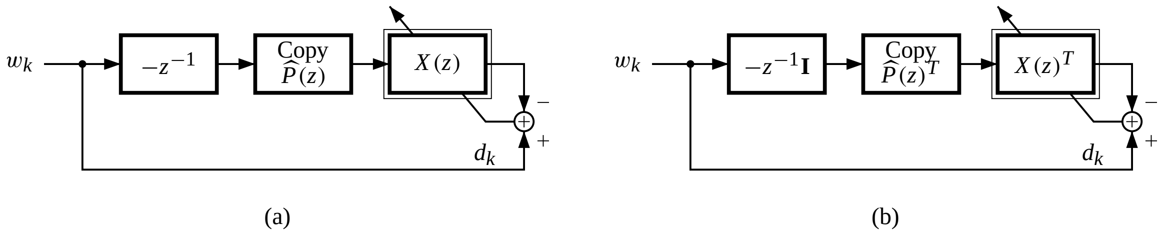


Fig. 5. Architectures for training disturbance canceler X; (a) for linear SISO systems; (b) for linear MIMO systems. The actual disturbance canceler is a digital copy of X.

B. *Linear SISO Disturbance Canceling*

The dynamic response of the system may now be controlled but plant disturbance, however, is not yet rejected. This can be accomplished via another special adaptive filter called the disturbance canceler X. In order to avoid bias in the adaptive plant model due to the disturbance, a special architecture must be used when adapting $\widehat{P}$ and X simultaneously. The solution is shown in Fig. 4. It can be shown that the plant model will adapt to the correct solution using this scheme [1,3] so long as the disturbance is is zero-mean and uncorrelated with u_k.

When considering the linear SISO case, we would like to adapt the disturbance canceler X such that $z^{-1}P(z)X(z) = -1$. This would entirely cancel the disturbance, but result in a non-causal X. We can still use this formula to adapt $X(z)$, as shown in Figure 5(a), but by the adaptive method, $X(z)$ will adapt to the optimal causal solution.[3] As when adapting a controller, we take advantage of the fact that linear systems commute in order to generate an adaptation signal for X.

IV. Linear MIMO Adaptive Inverse Control

A. *Feedforward Control*

While linear SISO systems have transfer functions, linear MIMO systems have transfer function matrices, denoted for example as $[P(z)]$. Matrix multiplication is not in general commutative; therefore,

[3] We note that this solution amounts to a predictor which predicts a future sample of the disturbance, cascaded with a plant inverse. Since optimal prediction is in general a nonlinear operation, the ideal disturbance canceler is a nonlinear adaptive filter even if the plant is linear!

$[h_{11}]_0$	$[h_{12}]_0$	$[h_{11}]_1$	$[h_{12}]_1$	$[h_{11}]_2$	$[h_{12}]_3$	$[h_{11}]_3$	$[h_{12}]_3$	$[h_{11}]_4$	$[h_{12}]_4$
$[h_{21}]_0$	$[h_{22}]_0$	$[h_{21}]_1$	$[h_{22}]_1$	$[h_{21}]_2$	$[h_{22}]_2$	$[h_{21}]_3$	$[h_{22}]_3$	$[h_{21}]_4$	$[h_{22}]_4$

Fig. 6. Weight matrix for filter $[H(z)]$.

$[P(z)][C(z)] \neq [C(z)][P(z)]$ and we can not use the same simple method as with the SISO system. A number of ways have been proposed to adapt a controller for a linear MIMO system [1–6]. For example,

- The "algebraic method" of reference [6] does not work when the plant has differing numbers of inputs and outputs. This method is also more cumbersome than the one to be developed.
- The "BPTM" method of references [2, 3] is extremely slow to converge.
- The "filtered-ϵ" method of reference [6] seems to have numeric difficulties, and often has trouble converging.

Here, we present a very simple and fast method to adapt a linear MIMO controller. It uses the fact that $[P(z)][C(z)] = [C(z)]^T[P(z)]^T$. The block diagram in Figure 3(b) may then be used to adapt the controller. The entire operation depends on being able to take the "transpose" of an adaptive filter representing a transfer function matrix. These filters are actually stored as impulse-response matrices, and the transpose operation is simply a re-organization of the components of the impulse-response matrices. The mechanics of taking a filter transpose are discussed later in Section C. As with the SISO linear case, convergence using the RLS algorithm occurs within twice as many iterations as there are taps in the longest impulse response in the MIMO controller filter C. This is an improvement of many orders of magnitude when compared with the other known cited methods.

B. Disturbance Canceling

We can adapt a disturbance canceler for a linear MIMO system using the same equation as for the SISO case, and adopting the transpose method used when adapting C for the MIMO system. The block diagram of the adaptation method is shown in Figure 5(b).

C. Transpose of Transfer Functions

In order to train the controller, notice that we need the transposed filter $[\hat{P}(z)]^T$. The weight matrix for this filter is not the same as $[\mathbf{W}_{\hat{P}}]^T$. To find the correct weight matrix for an arbitrary transposed filter $[H(z)]^T$ consider first the weight matrix for some arbitrary filter $[H(z)]$. For the sake of example we will assume $[H(z)]$ has two inputs, two outputs and $N = 4$ so that the impulse response is five samples long. Then, the H-matrix has entries as shown in Fig. 6

There are four impulse responses embedded in this matrix: h_{11}, h_{12}, h_{21} and h_{22}. When we take the z-transform and make the transfer-function matrix for this filter, we get

$$[H(z)] = \begin{bmatrix} H_{11}(z) & H_{12}(z) \\ H_{21}(z) & H_{22}(z) \end{bmatrix} \quad \text{which has transpose} \quad [H(z)]^T = \begin{bmatrix} H_{11}(z) & H_{21}(z) \\ H_{12}(z) & H_{22}(z) \end{bmatrix}.$$

So, the transpose of the weight filter has entries as shown in Fig. 7 on the following page.

Computing the transpose of the filter involves a simple reordering of the weights in the weight matrix of the filter. The MATLAB code in Appendix E, for example, performs the transpose operation.

$[h_{11}]_0$	$[h_{21}]_0$	$[h_{11}]_1$	$[h_{21}]_1$	$[h_{11}]_2$	$[h_{21}]_3$	$[h_{11}]_3$	$[h_{21}]_3$	$[h_{11}]_4$	$[h_{21}]_4$
$[h_{12}]_0$	$[h_{22}]_0$	$[h_{12}]_1$	$[h_{22}]_1$	$[h_{12}]_2$	$[h_{22}]_2$	$[h_{12}]_3$	$[h_{22}]_3$	$[h_{12}]_4$	$[h_{22}]_4$

Fig. 7. Weight matrix for filter $[H(z)]^T$.

D. Wiener Solution

There are three cases to consider when computing the Wiener solution for the controller weight matrix when the plant is linear MIMO. The first is when the plant has more outputs than inputs. The second is where the plant has fewer outputs than inputs. The third is when the plant has an equal number of outputs and inputs.

An important mathematical result relating to z-transforms of impulse response matrices must first be developed. Assume that white noise is filtered by a filter with transfer function $H(z)$. The z-transform of the autocorrelation of the filter output is then $\Phi_{yy}(z) = H(z^{-1})H(z)^T$. Since autocorrelation functions are symmetric, then we have the important results which hold for any $H(z)$

$$H(z^{-1})H(z)^T = H(z)H(z^{-1})^T$$

and (now letting the filter be $H(z)^T$)

$$H(z^{-1})^T H(z) = H(z)^T H(z^{-1}).$$

More outputs than inputs: If the plant has more outputs than inputs, then $\widehat{P}(z)^T \widehat{P}(z)$ is generally invertible. From the Wiener results in Appendix A and the block diagram in Fig. 3 on page 3 we have that the solution for the controller will be

$$\widehat{P}(z^{-1})^T \Phi_{nn}(z)\widehat{P}(z)C(z) = \widehat{P}(z^{-1})^T \Phi_{nn}(z)M(z).$$

Multiply on the left by $\left[\widehat{P}(z^{-1})^T \widehat{P}(z^{-1})\right]^{-1} \widehat{P}(z^{-1})^T \widehat{P}(z)$. Assume that the modeling noise is white such that $\Phi_{nn}(z) = I$. Then,

$$\left[\widehat{P}(z^{-1})^T \widehat{P}(z^{-1})\right]^{-1} \widehat{P}(z^{-1})^T \widehat{P}(z)\widehat{P}(z^{-1})^T \widehat{P}(z)C(z) = \left[\widehat{P}(z^{-1})^T \widehat{P}(z^{-1})\right]^{-1} \widehat{P}(z^{-1})^T \widehat{P}(z)\widehat{P}(z^{-1})^T M(z).$$

Using the identities above, we can replace $\widehat{P}(z)\widehat{P}(z^{-1})^T$ with $\widehat{P}(z^{-1})\widehat{P}(z)^T$ on the left- and right-hand sides. When terms cancel we are left with

$$\begin{aligned} \widehat{P}(z)^T \widehat{P}(z)C(z) &= \widehat{P}(z)^T M(z) \\ C^{(opt)}(z) &= \left[\widehat{P}(z)^T \widehat{P}(z)\right]^{-1} \widehat{P}(z)^T M(z), \end{aligned}$$

which is the pseudo-inverse of $\widehat{P}(z)$ in cascade with the reference model. This is the desired solution (although it would be better if the solution had $[P(z)^T P(z)]^{-1} P(z)^T M(z)$ instead).

More inputs than outputs: If the plant has more inputs than outputs, then some matrix cancellations in the above method will no longer be valid. Instead, $\widehat{P}(z)\widehat{P}(z)^T$ will generally be invertible. Again, we must start with

$$\widehat{P}(z^{-1})^T \Phi_{nn}(z)\widehat{P}(z)C(z) = \widehat{P}(z^{-1})^T \Phi_{nn}(z)M(z).$$

Assume that $\Phi_{nn}(z) = I$. Multiply on the left by $\left[\widehat{P}(z^{-1})\widehat{P}(z^{-1})^T\right]^{-1}\widehat{P}(z^{-1})$. We are left with

$$\widehat{P}(z)C(z) = M(z).$$

We notice (by substitution) that one possible solution for $C(z)$ is

$$C^{(opt)}(z) = \widehat{P}(z)^T\left[\widehat{P}(z)\widehat{P}(z)^T\right]^{-1} M(z).$$

This is the minimum-norm solution. Generally, when there are more inputs than outputs there will be many solutions to $C(z)$ which all achieve zero error. The minimum-norm solution is one of these, and uses the least amount of control effort (in a mean-squared sense). It is not a guaranteed solution, however.
Equal number of inputs and outputs: If the plant has an equal number of inputs and outputs, and if the plant is invertible, then both solutions become

$$C^{(opt)}(z) = \widehat{P}(z)^{-1}M(z).$$

We note that all three Wiener solutions are based on the inverse of the plant model and not on the inverse of the plant. So, the efficacy of this method is dependent on an accurate plant model. Research must be done to see if it can be extended, to be independent of an accurate plant model.

E. Matrix RLS

The *Recursive Least Squares* (RLS) algorithm is a fast and efficient method for adapting the weights of an adaptive linear filter [10]. Many other fast algorithms are available, and some posses better numerical and convergence properties [12]. Many are as fast as RLS, but none appear to be faster in terms of number of training iterations required to achieve convergence. So, for the sake of example, a form of RLS called matrix-RLS is used here. Matrix-RLS is used to adapt a linear MIMO filter, whereas regular RLS adapts a SISO filter.

Matrix-RLS may be derived in the same way as standard RLS [10]. The steps are omitted here. The final form of a numerically-stable version of the algorithm is shown in Algorithm 1.

Algorithm 1 Matrix-RLS algorithm.

$$\begin{aligned}
\pi(k) &= X(k)^T\Phi_{xx}^{-1}(k-1)\\
r(k) &= \frac{1}{\lambda + \pi(k)X(k)}\\
K(k) &= r(k)\pi(k)\\
\zeta(k) &= d(k) - \mathbf{W}(k-1)X(k)\\
\mathbf{W}(k) &= \mathbf{W}(k-1) + \zeta(k)K(k)\\
\Phi_{xx}^{-1}(k) &= \frac{1}{\lambda}\text{tri}\left\{\Phi_{xx}^{-1}(k-1) - \pi(k)^T K(k)\right\},
\end{aligned}$$

where the tri() operator takes the upper or lower triangular part of a matrix and replicates it in the lower or upper part to preserve symmetry. λ is a forgetting constant, and is set slightly less than 1. $X(k)$ is the tap-delay-line input to the filter at time k, such that the filter output is $\mathbf{W}(k)X(k)$. $\Phi_{xx}^{-1}(k)$ is initialized to a diagonal matrix with large (order of 10,000) entries.

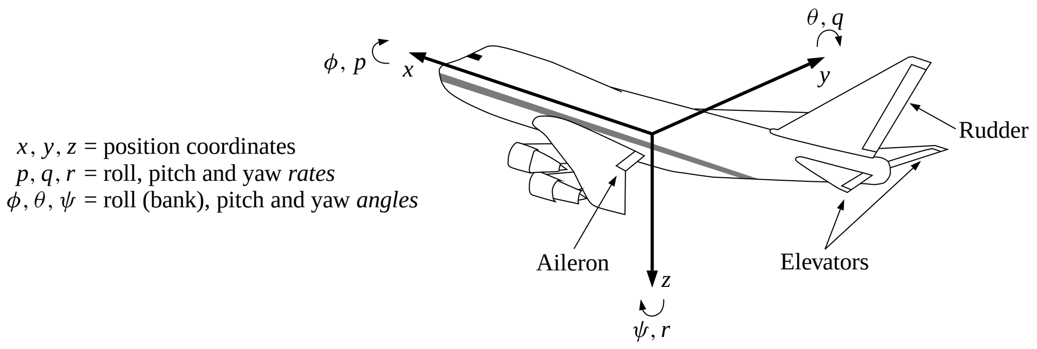

x, y, z = position coordinates
p, q, r = roll, pitch and yaw *rates*
ϕ, θ, ψ = roll (bank), pitch and yaw *angles*
ϕ, p
x
θ, q
y
Rudder
Aileron
Elevators
z
ψ, r

Fig. 8. Aircraft yaw-rate and bank-angle control.

F. Convergence

There are two convergence issues to deal with. First, convergence of the plant model, and then convergence of the controller/disturbance canceler. The plant model needs to converge before either the controller or disturbance canceler can converge. Here we assume that matrix-RLS is used to adapt the two filters.

According to Haykin, RLS converges in two times as many iterations as there are taps in the FIR filter [10]. So, for example, if a matrix-FIR filter has five taps per sub-filter, convergence is achieved in about ten iterations. Similarly, convergence of the controller or disturbance canceler occurs in about twice as many iterations as there are taps in each filter. However, since the controller- and canceler adaptation process is done offline, it can be done quickly in the background and entire system convergence occurs can occur in about twice as many iterations as there are taps in the plant-model FIR filter.

A note re. using matrix-LMS. The convergence speed of matrix-LMS is related to the eigenvalue spread of the autocorrelation matrix R of the filter input. The eigenvalue spread in the R matrix in the example presented next is fourteen orders of magnitude! Matrix-LMS converges very slowly.

V. Example

Two aspects of flight control for a Boeing 747 aircraft (see Figure 8) were selected to demonstrate linear, MIMO control.[4] The dynamics of the airplane have been approximated by a linear model around an equilibrium point. In the case at hand, the equilibrium "point" is level flight at 40,000 ft and a nominal forward speed of Mach 0.8 (774 ft/sec). The resulting linearized equations of motion are eighth-order, but they may be separated into two fourth-order sets representing the perturbations in longitudinal and lateral motion. Here, we wish to control the aircraft's yaw-rate (r) and bank-angle (ϕ).

The dynamics of the system are most compactly represented in state-space form. When converted to discrete-time, we define

$$u_k = \begin{bmatrix} \text{Rudder angle in degrees} \\ \text{Aileron angle in degrees} \end{bmatrix}, \qquad y_k = \begin{bmatrix} \text{Yaw rate, } r_k, \text{ in radians/second} \\ \text{Bank angle, } \phi_k, \text{ in radians} \end{bmatrix},$$

and,

$$x_k = \begin{bmatrix} \text{Sideslip angle, } \beta_k, \text{ in radians} \\ \text{Yaw rate, } r_k, \text{ in radians/second} \\ \text{Roll rate, } p_k, \text{ in radians/second} \\ \text{Bank angle, } \phi_k, \text{ in radians} \end{bmatrix}.$$

[4] The primary reference for this section is [13, pp. 684–93]. The author in turn references the seminal but elusive source [14]. The augmented equations for MIMO control were obtained from [15, pp. 23–35].

Then,

$$\begin{aligned} x_{k+1} &= A_d x_k + B_d u_k \\ y_k &= C_d x_k, \end{aligned}$$

where,

$$A_d = \begin{bmatrix} 0.8876 & -0.3081 & 0.0415 & 0.0198 \\ 0.2020 & 0.3973 & -0.0046 & 0.0024 \\ -1.2515 & 0.5106 & 0.7617 & -0.0139 \\ -0.3313 & 0.1510 & 0.4407 & 0.9976 \end{bmatrix}, \qquad B_d = \begin{bmatrix} 0.4806 & -0.0013 \\ -1.5809 & 0.3887 \\ 0.0599 & 4.8390 \\ 0.0390 & 1.2585 \end{bmatrix}.$$

The output matrix C_d was chosen to be either

$$C_d = \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix}$$

$$C_d = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$C_d = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

depending on whether the plant had 1, 2 or 3 outputs.

The reference command to be tracked is generated independently for each output. The reference command for the desired yaw-rate is a first-order Markov process generated by filtering i.i.d. uniform random variables with maximum value 0.03 using a one-pole filter whose pole is at $z = 0.9$. The reference command for the desired bank angle is a first-order Markov process generated by filtering i.i.d. uniform random variables with maximum value 0.12 with a one-pole filter whose pole is at $z = 0.9$.

The primary disturbance experienced by the dynamics of the airplane are those induced by bursts of wind. It is assumed here that the nominal wind values are incorporated into the dynamic model of flight, and that gusts around that nominal value are the disturbances. The state of the airplane, x_k, is affected directly by the wind. So, the full discrete-time model of the airplane dynamics, with disturbance, is

$$\begin{aligned} x_{k+1} &= A_d\,[x_k + \mathrm{dist}_k] + B_d u_k \\ y_k &= C_d x_k, \end{aligned}$$

Furthermore, it is assumed that the wind gusts occur as planar fronts and thus do not affect the yaw-rate, roll-rate or bank-angle directly. Instead, the sideslip angle is directly affected by the wind, and the other state-variables are affected indirectly through the dynamical relationship between themselves and the sideslip angle. If we model the wind in the lateral direction, then the sideslip angle is perturbed by

$$\Delta\beta_k = \mathrm{atan}\left(\frac{\text{wind speed}}{\text{airplane speed}}\right).$$

The model for generating a wind-speed time series is based on data presented in reference [16]. An approximation is made to the autocorrelation function of the cited paper. The power spectral density of wind velocity was calculated from the autocorrelation function, and was found to be

$$\Phi(f) = \frac{3950}{1 + (20\pi f)^2}.$$

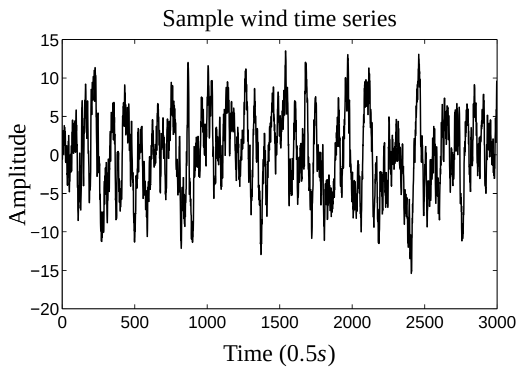


Fig. 9. Sample wind time series.

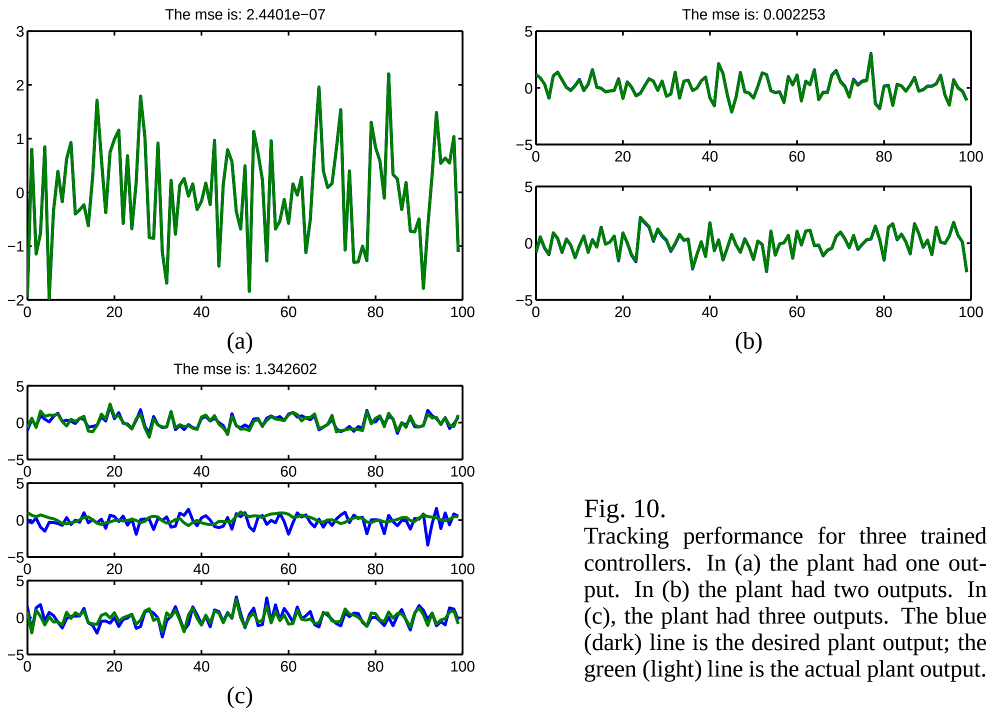


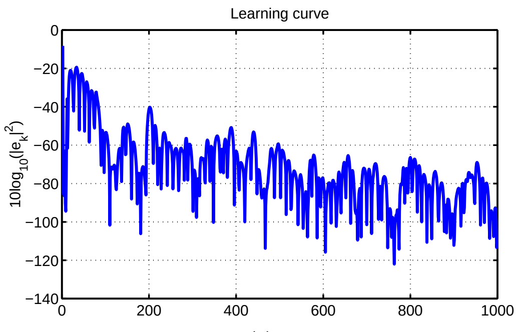
Fig. 10.
Tracking performance for three trained controllers. In (a) the plant had one output. In (b) the plant had two outputs. In (c), the plant had three outputs. The blue (dark) line is the desired plant output; the green (light) line is the actual plant output.

An FIR filter was designed using a weighted least-squares optimization algorithm to produce this power spectral density given an input stream of i.i.d. uniform random numbers with maximum magnitude 1. A sample wind time-sequence is shown in Figure 9. The maximum absolute wind speed is in the neighborhood of 20 feet per second, so the maximum perturbation to β_k is around 0.03 radians.

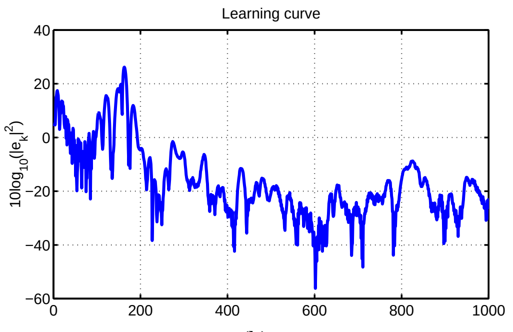
Note: simulations in reference [3] considered two-input two-output control. The controller, trained with the BPTM method, converged in about 10^8 iterations. The present method converges in about 200–400 iterations!

Tracking results (in the absence of added disturbance) for the three cases simulated are presented in Fig. 10. Tracking is essentially perfect when there are more plant inputs than outputs, and is also very good when there are an equal number of plant inputs and outputs. When there are more plant outputs than inputs, it becomes impossible to get perfect tracking in general, and the controller adapts to find the solution which minimizes the mean-square output error.

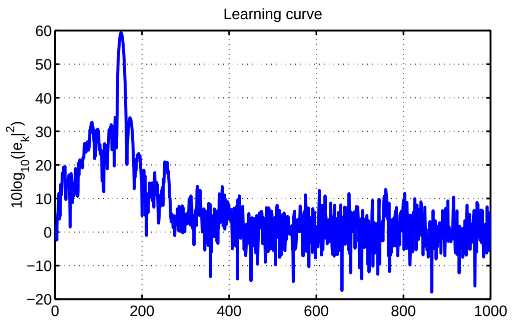
Learning curves for the three cases are presented in Fig. 11 on the following page. We see convergence in all cases within 200–400 iterations.



(a)



(b)



(c)

Fig. 11.
Learning curves training the three controllers (in dB). In (a) the plant had one output. In (b) the plant had two outputs. In (c), the plant had three outputs. The horizontal axis is in iterations.

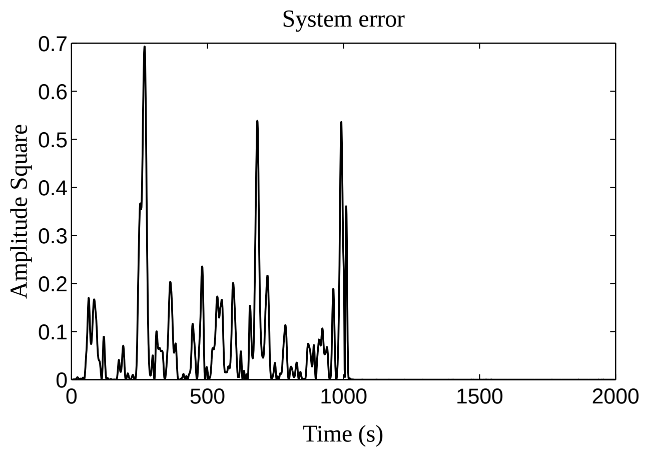


Fig. 12. Disturbance canceling for MIMO system.

A. *Disturbance Canceling*

Disturbance canceling was tested for the MIMO system, and results are plotted in Fig. 12. The figure shows the squared-norm of the system error plotted versus iteration. Disturbance was present at all times, and the (trained) disturbance canceler was turned “on” at time 1000. As can be seen, the disturbance canceler removes essentially all of the disturbance, resulting in near-perfect tracking even in the presence of disturbance.

VI. Conclusions

The new method presented in this report for adapting controllers and disturbance cancelers for linear MIMO plants works very well and is very fast. Items on the “wish list” include: Finding a way to make the controller unbiased by plant modeling errors, and adding constraints to the control effort produced by the controller.

APPENDICES

A. MATRIX MIMO WIENER SOLUTION

The Wiener solution for adaptive filtering when the filter is a linear MIMO system is outlined in this appendix. First, the unconstrained solution is presented, followed by the causal solution and an application to a common structure.

A. The Unconstrained MIMO Wiener Solution

The two-sided Wiener solution is developed here for a MIMO filter. The input to the filter is the vector signal $[x_k]$ and the output of the filter is the vector signal $\left[y_k\right]$. The desired response must have the same dimension as $\left[y_k\right]$; it is labeled $[d_k]$.

The MIMO filter has a transfer-function matrix, such that (for an example 2×2 system)

$$\begin{bmatrix} Y_1(z) \\ Y_2(z) \end{bmatrix} = \begin{bmatrix} H_{11}(z) & H_{12}(z) \\ H_{21}(z) & H_{22}(z) \end{bmatrix} \begin{bmatrix} X_1(z) \\ X_2(z) \end{bmatrix}.$$

In other words,

$$\begin{aligned} Y_1(z) &= H_{11}(z)X_1(z) + H_{12}(z)X_2(z) \\ Y_2(z) &= H_{21}(z)X_1(z) + H_{22}(z)X_2(z). \end{aligned}$$

In the time domain, we have

$$\begin{aligned} y_1[k] &= h_{11}[k] * x_1[k] + h_{12}[k] * x_2[k] \\ y_2[k] &= h_{21}[k] * x_1[k] + h_{22}[k] * x_2[k]. \end{aligned}$$

This can be written as using summations

$$\begin{aligned} y_1[k] &= \sum_{m=-\infty}^{\infty} h_{11}[m]x_1[k-m] + \sum_{m=-\infty}^{\infty} h_{12}[m]x_2[k-m] \\ y_2[k] &= \sum_{m=-\infty}^{\infty} h_{21}[m]x_1[k-m] + \sum_{m=-\infty}^{\infty} h_{22}[m]x_2[k-m]. \end{aligned}$$

The summations may be combined such that

$$\begin{aligned} y_1[k] &= \sum_{m=-\infty}^{\infty} \left(h_{11}[m]x_1[k-m] + h_{12}[m]x_2[k-m]\right) \\ y_2[k] &= \sum_{m=-\infty}^{\infty} \left(h_{21}[m]x_1[k-m] + h_{22}[m]x_2[k-m]\right). \end{aligned}$$

Furthermore, this can be written in matrix form: (the key to the whole proof!)

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix}_{[k]} = \sum_{m=-\infty}^{\infty} \begin{bmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{bmatrix}_{[m]} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_{[k-m]}$$

or

$$\left[y_k\right] = \sum_{m=-\infty}^{\infty} [h_m] \left[x_{k-m}\right].$$

Now for the Wiener proof. Note that $[e_k] = [d_k] - \left[y_k\right]$. We wish to minimize the expected 2-norm of $[e_k]$:

$$\begin{aligned}
\|[e_k]\|^2 &= [e_k]^T [e_k] \\
&= [d_k]^T [d_k] + \sum_{l=-\infty}^{\infty}\sum_{u=-\infty}^{\infty} \left([h_l]\left[x_{k-l}\right]\right)^T [h_u]\left[x_{k-u}\right] - 2\sum_{l=-\infty}^{\infty} \left([h_l]\left[x_{k-l}\right]\right)^T [d_k] \\
&= [d_k]^T [d_k] + \sum_{l=-\infty}^{\infty}\sum_{u=-\infty}^{\infty} \left[x_{k-l}\right]^T [h_l]^T [h_u]\left[x_{k-u}\right] - 2\sum_{l=-\infty}^{\infty} \left[x_{k-l}\right]^T [h_l]^T [d_k].
\end{aligned}$$

Each composite quantity is a scalar, so $() = Tr()$. Furthermore $Tr(AB) = Tr(BA)$ for all $A \in C^{n\times m}$ and $B \in C^{m\times n}$.

$$\begin{aligned}
\|[e_k]\|^2 &= [d_k]^T [d_k] + \sum_{l=-\infty}^{\infty}\sum_{u=-\infty}^{\infty} Tr\left([h_l]^T [h_u]\left[x_{k-u}\right]\left[x_{k-l}\right]^T\right) - 2\sum_{l=-\infty}^{\infty} Tr\left([h_l]^T [d_k]\left[x_{k-l}\right]^T\right) \\
\mathbb{E}\left(\|[e_k]\|^2\right) &= \mathbb{E}\left([d_k]^T [d_k]\right) + \sum_{l=-\infty}^{\infty}\sum_{u=-\infty}^{\infty} Tr\left([h_l]^T [h_u]\mathbb{E}\left(\left[x_{k-u}\right]\left[x_{k-l}\right]^T\right)\right) \\
&\quad -2\sum_{l=-\infty}^{\infty} Tr\left([h_l]^T \mathbb{E}\left(\underbrace{[d_k]\left[x_{k-l}\right]^T}_{\phi_{dx}(-l)=\phi_{xd}(l)^T}\right)\right) \\
&= Tr\left([\phi_{dd}(0)]\right) + \sum_{l=-\infty}^{\infty}\sum_{u=-\infty}^{\infty} Tr\left([h_l]^T [h_u][\phi_{xx}(l-u)]\right) - 2\sum_{l=-\infty}^{\infty} Tr\left([h_l]^T [\phi_{xd}(l)]^T\right).
\end{aligned}$$

To find the optimal filter weights, we take the derivative of $\mathbb{E}\left(\|e_k\|^2\right)$ and set it to zero. The following identities help:

$$\begin{aligned}
\frac{\partial Tr(AX^TB)}{\partial X} &= BA \\
\frac{\partial Tr(AXB)}{\partial X} &= A^TB^T \\
\frac{\partial Tr(AXBX^T)}{\partial X} &= A^TXB^T + AXB.
\end{aligned}$$

So,

$$\frac{\partial \mathbb{E}\left(\|[e_k]\|^2\right)}{\partial h_j} = 0 + \frac{\partial(\sum\sum)}{\partial h_j} - 2\frac{\partial(\sum)}{\partial h_j} \equiv 0.$$

Now,

$$\frac{\partial(\sum)}{\partial h_j} = \frac{\partial Tr\left(\left[h_j\right]^T\left[\phi_{xd}(j)\right]^T\right)}{\partial h_j} = \left[\phi_{xd}(j)\right]^T$$

since all of the summation terms not containing $\left[h_j\right]$ go away under the differentiation, and we let $A = I$ and $B = \left[\phi_{xd}(j)\right]^T$. Also, the remaining partial may be computed for different values of summation indices. If $l \neq j$ and $u \neq j$

$$\frac{\partial(\sum\sum)}{\partial h_j} = 0.$$

If $l = j$ and $u \neq j$ then

$$\frac{\partial(\sum\sum)}{\partial h_j} = \frac{\partial Tr\left(\left[h_j\right]^T [h_u]\left[\phi_{xx}(j-u)\right]\right)}{\partial h_j} = [h_u]\left[\phi_{xx}(j-u)\right].$$

If $l \neq j$ and $u = j$ then

$$\frac{\partial(\sum\sum)}{\partial h_j} = \frac{\partial Tr\left([h_l]^T \left[h_j\right]\left[\phi_{xx}(l-j)\right]\right)}{\partial h_j} = [h_l]\left[\phi_{xx}(l-j)\right]^T = [h_l]\left[\phi_{xx}(j-l)\right].$$

If $l = j$ and $u = j$ then

$$\frac{\partial(\sum\sum)}{\partial h_j} = \frac{\partial Tr\left(\left[h_j\right]^T \left[h_j\right][\phi_{xx}(0)]\right)}{\partial h_j} = \frac{\partial Tr\left(\left[h_j\right][\phi_{xx}(0)]\left[h_j\right]^T\right)}{\partial h_j} = 2\,[h_l]\,[\phi_{xx}(0)].$$

Putting all of the above together, we have

$$\frac{\partial \mathbb{E}\left(\|[e_k]\|^2\right)}{\partial h_j} = 2\sum_{l=-\infty}^{\infty}[h_l]\left[\phi_{xx}(j-l)\right] - 2\left[\phi_{xd}(j)\right]^T \equiv 0.$$

Take z-transforms of the above, realizing that the summation is a convolution,

$$\begin{aligned}[H(z)]\,[\Phi_{xx}(z)] &= [\Phi_{xd}(z)]^T \\ [H(z)] &= [\Phi_{xd}(z)]^T\,[\Phi_{xx}(z)]^{-1}.\end{aligned}$$

B. The Causal MIMO Wiener Solution

In order to use the unconstrained Wiener filter solution to find the causal Wiener filter solution, we can use the causal part of the unconstrained solution if we first “whiten” the input to that causal filter.

We want to whiten x_k and to do so we suppose that x_k was generated via filtering white noise through the filter $[G(z)]$. Then, (using the autocorrelation result in the next section), we can write

$$\Phi_{xx}(z) = [G(z^{-1})][I][G(z)]^T = [G(z^{-1})][G(z)]^T.$$

Let the whitening filter be $H_w(z) = [G(z)]^{-1}$. Note that if $[G(z)]$ is minimum-phase,

$$\begin{aligned}\Phi^+_{xx}(z) &= [G(z)] \\ \Phi^-_{xx}(z) &= [G(z^{-1})].\end{aligned}$$

and $\Phi_{xx}(z) = \Phi^-_{xx}(z)\Phi^+_{xx}(z)^T$. Now, consider this in the system of Fig. 13 on the following page. Computing the necessary correlations (see next section), we get $\Phi_{zd}(z) = [\Phi^+_{xx}(z^{-1})]^{-1}[I][\Phi_{xd}(z)]$. Since $z(z)$ is white, then $\Phi_{zz}(z) = I$. Then, the unconstrained Wiener solution for $H_c(z)$ is $[\Phi_{xd}(z)]^T[\Phi^-_{xx}(z)]^{-1}$. The constrained Wiener solution is the causal part of $H_c(z)$ preceded by the whitening filter, or

$$[H(z)]_{\text{causal}} = \left[[\Phi_{xd}(z)]^T[\Phi^-_{xx}(z)]^{-1}\right]_+\left[\Phi^+_{xx}(z)\right]^{-1}.$$

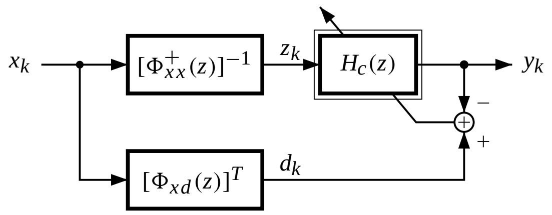


Fig. 13. System for computing causal Wiener solution.

C. *Computing* $\Phi_{xd}(z)$ *and* $\Phi_{xx}(z)$.

Many times, we must compute the quantities $\Phi_{xd}(z)$ and $\Phi_{xx}(z)$ for block diagrams of the form of Fig. 14 on the next page. To find $\Phi_{xd}(z)$ and $\Phi_{xx}(z)$ we first compute $\phi_{xd}(m)$.

$$\begin{aligned}
\phi_{xd}(m) &= \mathbb{E}[x_k d^T_{k+m}] \\
x_k &= \sum_{l=-\infty}^{\infty} \alpha_l n_{k-l} \\
d_k &= \sum_{i=-\infty}^{\infty} \beta_i n_{k-i} \\
\phi_{xd}(m) &= \mathbb{E}\left[\sum_{l=-\infty}^{\infty}\sum_{i=-\infty}^{\infty} \alpha_l n_{k-l} n^T_{k+m-i}\beta_i^T\right] \\
&= \sum_{l=-\infty}^{\infty}\sum_{i=-\infty}^{\infty} \alpha_l \phi_{nn}(m+l-i)\beta_i^T.
\end{aligned}$$

This may be unwrapped by taking the z-transform.

$$\begin{aligned}
\Phi_{xd}(z) &= \sum_{l=-\infty}^{\infty} \alpha_l z^{+l} \sum_{m=-\infty}^{\infty}\sum_{i=-\infty}^{\infty} \phi_{nn}(m+l-i) z^{-(m+l-i)} \beta_i^T z^{-i} \\
&= \sum_{l=-\infty}^{\infty} [\alpha_l z^{+l}] \left[\sum_{i=-\infty}^{\infty} [\beta_i z^{-i}] \sum_{m=-\infty}^{\infty} \phi^T_{nn}(m+l-i) z^{-(m+l-i)}\right]^T \\
&= \sum_{l=-\infty}^{\infty} [\alpha_l z^{+l}] \left[B(z)\Phi^T_{nn}(z)\right]^T \\
&= A(z^{-1})\Phi_{nn}(z)B(z)^T.
\end{aligned}$$

By extension, $\Phi_{xx}(z) = A(z^{-1})\Phi_{nn}(z)A(z)^T$.

B. Main Simulation Code

```
% ================================================================
% Simple method for adapting a MIMO linear controller using adaptive
% inverse control.  Copyright (c) 2000.  Dr.  Gregory L. Plett.  Please
% distribute with this copyright message.
```

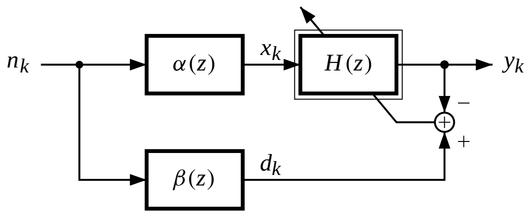


Fig. 14. Generic adaptive structure.

```
% ======================================================================
clf;
numpin=2;               % number of plant inputs
numpout=1;              % number of plant outputs
numptap=61;             % number of taps in plant model
plantsetup;
reporttype=1;           % 1=tracking progress; 2=impulse responses
report=1000;            % frequency of reports

maxiter=1000;           % how many adaptive iterations.  Convergence ~ 200.
dksave=zeros([numpout report]);
yksave=zeros([numpout report]);
eksave=zeros([1 report]);

WmT=[zeros(numpout) eye(numpout)];      % ref-model trans---a unit delay
MTin=zeros([length(WmT(1,:))  1]);       % the input to M-transpose
Min=zeros([length(WmT(1,:))  1]);        % the input to M

Pin=zeros([numpin*numptap 1]);  % the input to the true plant
PHin=Pin;                       % the input to Phat to adapt C-transpose
PHTin=zeros([numpout*numptap 1]);% The input to Phat-transpose to adapt C-trans

numctap=71;
CCin=zeros([numpout*numctap 1]);% the input to C-copy
CTin=zeros([numpin*numctap 1]); % the input to C-transpose
WcT=zeros([numpout numpin*numctap]);% the initl weights of the ctrlr-transpose

lambda=0.9999;                  % the RLS forgetting factor
PCT=inv(0.0001)*eye(length(CTin));% The initial RLS inverse cov.  matrix for C

i=1;
mse=0;
for iter=1:maxiter,
  % get system desired response
  ik=randn([numpout 1]);        % random input to system
  PHTin=[ik; PHTin(1:end-length(ik),1)];
  xk=Wpht*PHTin;
  CTin=[xk; CTin(1:end-length(xk),1)];

  MTin=[ik; MTin(1:end-length(ik),1)];% shift in to M-transpose
  dkt=WmT*MTin;                 % desired output

  % update weights in C-transpose via matrix RLS
  pin=CTin'*PCT;
  rn=1/(lambda+pin*CTin);
  kn=rn*pin;
  chin=dkt-WcT*CTin;
  WcT=WcT+chin*kn;
  PCT=(PCT-pin'*kn)/lambda;
  PCT=triu(PCT)+triu(PCT,1)';

  Wc=ftranspose(WcT,numctap);

  % Feedforward system to plot tracking
  nk=randn([numpout 1]);               % random input to system
  CCin=[nk; CCin(1:end-length(nk),1)];
  uk=Wc*CCin;
  Pin=[uk; Pin(1:end-length(uk),1)];
  yk=Wp*Pin;

  yksave(:,i)=yk;
  Min=[nk; Min(1:end-length(nk),1)];
  dk=WmT*Min;
  dksave(:,i)=dk;
  % add plant noise here
  yksave(:,i)=yk;
  eksys=dk-yk; if iter>500, mse=mse+eksys'*eksys; end;
  eksave(i)=eksys'*eksys;

  i=i+1;
  % display impulse-response graphs every 50 iterations
  if (mod(iter,report)==0),
    plotreport;
    i=1;
  end;
end;
mse/(maxiter-500)
```

C. Plant Setup Code

```
% ====================================================================
% Sets up the MIMO plant
% Copyright (c) 2000.  Dr.  Gregory L. Plett.  Please distribute
% with this copyright message.
% ====================================================================
A=[0.88763 -0.30806  0.04148    0.019846
   0.20197  0.39735 -0.0046012  0.0024034
  -1.2515   0.5106   0.76171   -0.013941
  -0.33126  0.15104  0.44069    0.99763];
B=[0.48062 -0.0012813
  -1.5809   0.38869
   0.059924 4.839
   0.038972 1.2585];
% Compute impulse-responses assuming two-input, three-output and use
% only the impulse-responses we need.
C=[0 1 0 0; 0 0 0 1; 0 0 1 0];
D=[0 0; 0 0; 0 0];
sysd=ss(A,B,C,D,0.5);   % create state-space discrete-time system, T=0.5
H=impulse(sysd,(numptap-1)/2);% get impulse responses
h11=H(:,1,1)'; h12=H(:,1,2)'; % form individual impulse responses
h21=H(:,2,1)'; h22=H(:,2,2)';
h31=H(:,3,1)'; h32=H(:,3,2)';
```

```
% create the impulse-response filter matrix
Wph=zeros([numpout numpin*numptap]);
Wph(1,1:numpin:numpin*numptap-1)=h11;
Wph(1,2:numpin:numpin*numptap)=h12;
if numpout>1,
  Wph(2,1:numpin:numpin*numptap-1)=h21; Wph(2,2:numpin:numpin*numptap)=h22;
  if numpout>2,
   Wph(3,1:numpin:numpin*numptap-1)=h31; Wph(3,2:numpin:numpin*numptap)=h32;
  end;
end;
Wp=Wph;         % true plant
% create the transpose impulse-response filter
Wpht=ftranspose(Wph,numptap);
```

D. Report Plotting Code

```
% ==============================================================
% Plots reports for the simulations
% Copyright (c) 2000.  Dr.  Gregory L. Plett.  Please distribute
% with this copyright message.
% ==============================================================
if reporttype==1, % tracking
  themse=sum(sum((dksave-yksave).^2))/report;
  themse=sprintf('The mse is:  %f',themse);
  subplot(numpout,1,1);
  plot(0:report-1,dksave(1,:),0:report-1,yksave(1,:));
  title(themse);
  if numpout>1,
    subplot(numpout,1,2);
    plot(0:report-1,dksave(2,:),0:report-1,yksave(2,:));
    if numpout>2,
      subplot(numpout,1,3);
      plot(0:report-1,dksave(3,:),0:report-1,yksave(3,:));
     end;
  end;
elseif reporttype==2, % impulse responses
  wctranspose=ftranspose(Wc,numctap);
  c11=wctranspose(1,1:2:end-1); c21=wctranspose(1,2:2:end);
  I11=conv(h11,c11)+conv(h12,c21);
  if numpout>1,
    c12=wctranspose(2,1:2:end-1); c22=wctranspose(2,2:2:end);
    I12=conv(h11,c12)+conv(h12,c22);
    I21=conv(h21,c11)+conv(h22,c21);
    I22=conv(h21,c12)+conv(h22,c22);
    if numpout>2,
      c13=wctranspose(3,1:2:end-1); c23=wctranspose(3,2:2:end);
      I13=conv(h11,c13)+conv(h12,c23);
      I23=conv(h21,c13)+conv(h22,c23);
      I31=conv(h31,c11)+conv(h32,c21);
      I32=conv(h31,c12)+conv(h32,c22);
      I33=conv(h31,c13)+conv(h32,c23);
    end;
  end;
  subplot(numpout,numpout,1); stem(0:length(I11)-1,I11); title('I11');
  if numpout>1,
    subplot(numpout,numpout,2);
    stem(0:length(I12)-1,I12); title('I12');
    subplot(numpout,numpout,numpout+1);
    stem(0:length(I21)-1,I21); title('I21');
    subplot(numpout,numpout,numpout+2);
    stem(0:length(I22)-1,I22); title('I22');
    if numpout>2,
      subplot(numpout,numpout,3);
      stem(0:length(I13)-1,I13); title('I13');
      subplot(numpout,numpout,6);
      stem(0:length(I23)-1,I23); title('I23');
      subplot(numpout,numpout,7);
      stem(0:length(I31)-1,I31); title('I31');
      subplot(numpout,numpout,8);
      stem(0:length(I32)-1,I32); title('I32');
      subplot(numpout,numpout,9);
      stem(0:length(I33)-1,I33); title('I33');
    end;
  end;
elseif reporttype==3, % impulse responses of Wc
  wctranspose=ftranspose(Wc,numctap);
  c11=wctranspose(1,1:2:end-1); c21=wctranspose(1,2:2:end);
  if numpout>1,
    c12=wctranspose(2,1:2:end-1); c22=wctranspose(2,2:2:end);
    if numpout>2,
      c13=wctranspose(3,1:2:end-1); c23=wctranspose(3,2:2:end);
    end;
  end;
  subplot(2,numpout,1); stem(0:length(c11)-1,c11); title('c11');
  subplot(2,numpout,numpout+1); stem(0:length(c21)-1,c21); title('c21');
  if numpout>1,
    subplot(2,numpout,2); stem(0:length(c12)-1,c12); title('c12');
    subplot(2,numpout,numpout+2); stem(0:length(c22)-1,c22); title('c22');
    if numpout>2,
      subplot(2,numpout,3); stem(0:length(c13)-1,c13); title('c13');
      subplot(2,numpout,numpout+3); stem(0:length(c23)-1,c23); title('c23');
    end;
  end;
elseif reporttype==4, % impulse responses of Wv
  WvT=ftranspose(Wv,numctap);
  c11=WvT(1,1:2:end-1); c21=WvT(1,2:2:end);
  if numpout>1,
    c12=WvT(2,1:2:end-1); c22=WvT(2,2:2:end);
    if numpout>2,
      c13=WvT(3,1:2:end-1); c23=WvT(3,2:2:end);
    end;
  end;
  subplot(2,numpout,1); stem(0:length(c11)-1,c11); title('v11');
  subplot(2,numpout,numpout+1); stem(0:length(c21)-1,c21); title('v21');
  if numpout>1,
    subplot(2,numpout,2); stem(0:length(c12)-1,c12); title('v12');
    subplot(2,numpout,numpout+2); stem(0:length(c22)-1,c22); title('v22');
    if numpout>2,
      subplot(2,numpout,3); stem(0:length(c13)-1,c13); title('v13');
      subplot(2,numpout,numpout+3); stem(0:length(c23)-1,c23); title('v23');
    end;
  end;
```

```
end;
drawnow,
```

E. Filter Transpose Code

```
function fout=ftranspose(fin,numtaps);
% This function computes the filter transpose of the input
% MIMO linear filter.  The input filter is a matrix of
% dimension (numout)x(numin*numtaps).
%
% e.g.  The input filter has two outputs, three inputs and
% two taps.
%
%   +--------+--------+--------+--------+--------+--------+
%   | f11(1) | f12(1) | f13(1) | f11(2) | f12(2) | f13(2) |
%   +--------+--------+--------+--------+--------+--------+
%   | f21(1) | f22(1) | f23(1) | f21(2) | f22(2) | f23(2) |
%   +--------+--------+--------+--------+--------+--------+
%
% The algorithm first reshapes into a 3-D matrix:
%
%   +--------+--------+--------+    +--------+--------+--------+
%   | f11(1) | f12(1) | f13(1) |    | f11(2) | f12(2) | f13(2) |
%   +--------+--------+--------+    +--------+--------+--------+
%   | f21(1) | f22(1) | f23(1) |    | f21(2) | f22(2) | f23(2) |
%   +--------+--------+--------+    +--------+--------+--------+
%
% Then, it reorders the dimensions:
%
%   +--------+--------+    +--------+--------+
%   | f11(1) | f21(1) |    | f11(2) | f21(2) |
%   +--------+--------+    +--------+--------+
%   | f12(1) | f22(1) |    | f12(2) | f22(2) |
%   +--------+--------+    +--------+--------+
%   | f13(1) | f23(1) |    | f13(2) | f23(2) |
%   +--------+--------+    +--------+--------+
%
% Then it puts everything back together:
%
%   +--------+--------+--------+--------+
%   | f11(1) | f21(1) | f11(2) | f21(2) |
%   +--------+--------+--------+--------+
%   | f12(1) | f22(1) | f12(2) | f22(2) |
%   +--------+--------+--------+--------+
%   | f13(1) | f23(1) | f13(2) | f23(2) |
%   +--------+--------+--------+--------+
%
% Copyright (c) 2000.  Dr.  Gregory L. Plett.  Please distribute
% with this copyright message.
%
[numout numin]=size(fin); numin=numin/numtaps;
fout=reshape(permute(reshape(fin,[numout,numin,numtaps]), ...
     [2 1 3]), [numin numout*numtaps]);
```

REFERENCES

[1] G. L. Plett, “Adaptive inverse control of linear and nonlinear systems using dynamic neural networks”, *Neural Networks,* (submitted March 2000, under revision).

[2] G. L. Plett, “Adaptive inverse control of SISO and MIMO linear systems”, *International Journal of Adaptive Control and Signal Processing,* (submitted April 2000).

[3] G. L. Plett, *Adaptive Inverse Control of Plants with Disturbances,* PhD thesis, Stanford University, Stanford, CA 94305, May 1998.

[4] B. Widrow, G. L. Plett, E. Ferreira, and M. Lamego, “Adaptive inverse control based on nonlinear adaptive filtering”, in *Proceedings of 5th IFAC Workshop on Algorithms and Architectures for Real-Time Control AARTC’98,* (Cancun, MX: April 1998), pp. 247–252, (invited paper).

[5] B. Widrow and G. L. Plett, “Nonlinear adaptive inverse control”, in *Proceedings of the 36th IEEE Conference on Decision and Control,* (San Diego, CA: 10–12 December 1997), vol. 2, pp. 1032–1037.

[6] B. Widrow and E. Walach, *Adaptive Inverse Control,* Prentice Hall P T R, Upper Saddle River, NJ, 1996.

[7] B. Widrow and G. L. Plett, “‘Intelligent’ adaptive inverse control”, in *Proceedings of IFAC,* (San Francisco, CA: July 1996), pp. 104–105.

[8] B. Widrow and G. L. Plett, “Adaptive inverse control based on linear and nonlinear adaptive filtering”, in *Proceedings of International Workshop on Neural Networks for Identification, Control, Robotics and Signal/Image Processing,* (Venice, Italy: 21–23 August 1996), pp. 30–38.

[9] B. Widrow and G. L. Plett, “Adaptive inverse control based on linear and nonlinear adaptive filtering”, in *Proceedings of the World Congress on Neural Networks,* (San Diego, CA: September 1996), pp. 620–27.

[10] S. Haykin, *Adaptive Filter Theory,* Prentice Hall, Upper Saddle River, NJ, third edition, 1996.

[11] B. Widrow and S. D. Stearns, *Adaptive Signal Processing,* Prentice-Hall, Englewood Cliffs, NJ, 1985.

[12] G. Glentis, K Berberidis, and Sergios Theodoridis, “Efficient least squares adaptive algorithms for FIR transversal filtering”, *IEEE Signal Processing Magazine,* vol. 16, no. 4, pp. 13–41, July 1999.

[13] G. F. Franklin, J. D. Powell, and A. Emami-Naeini, *Feedback Control of Dynamic Systems,* Addison-Wesley, Reading, MA, third edition, 1994.

[14] R. K. Heffley and W. F. Jewell, “Aircraft handling qualities”, Tech. Rep. 1004-1, System Technology, Inc., Hawthorne, CA, May 1972.

[15] A. Grace, A. J. Laub, J. N. Little, and C. M. Thompson, *Control System Toolbox for use with MATLAB,* The Math Works Inc, Natick, MA, 1992.

[16] F. C. Kaminsky, R. H. Kirchhoff, C. Y. Syu, and J. F. Manwell, “A comparison of alternative approaches for the synthetic generation of a wind speed time series”, *Transactions of the American Society of Mechanical Engineers. Journal of Solar Energy Engineering,* vol. 113, no. 4, pp. 280–89, November 1991.