Design of Fault-Tolerant LTI State-Space Systems*

C. N. Hadjicostis† G. C. Verghese

Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology, Cambridge, MA 02139
chris@allegro.mit.edu verghese@mit.edu

Abstract

The design of linear time-invariant (LTI) systems in state form has traditionally focused on implementations that require the least number of state variables. Such *minimal* designs have attempted to limit the required resources, such as hardware, computation time or power, by minimizing the system dimension. In recent years, however, the increasing necessity for the design and implementation of fault-tolerant systems has proved that "controlled redundancy" (that is, redundancy that has been intentionally introduced in some systematic way) can be extremely important: it can be used to detect and correct errors or to guarantee desirable performance despite hardware or computational failures.

Modular redundancy, the traditional approach to fault tolerance, is prohibitively expensive because of the overhead in replicating the hardware. This paper discusses alternative methods for systematically introducing redundancy for LTI systems in state form. Our approach consists of mapping the state space of the original system into a redundant space of higher dimension while preserving, within this larger space, the properties of the original system in some encoded form. We provide a complete characterization of the class of appropriate redundant LTI systems and illustrate through several examples ways in which our framework can be used for achieving fault tolerance.

1 Introduction

In this paper we explore a design methodology for fault-tolerant linear time-invariant (LTI) systems in state form. Our approach is based on mapping the state of the original system into a larger, redundant space while at the same time preserving the properties and information contained in the original system — perhaps in some encoded form. The redundancy we add into the system can be used to achieve error correction or robust performance despite hardware failures.

Traditional system design has aimed at the realization of *minimal* systems, i.e., systems that require minimal resources (these resources could be hardware, computation time, power consumption, system dimension, etc.). Recently, however, there has been an increasing interest in redundant systems that are *fault-tolerant*. The traditional, but rather inefficient, way of designing fault-tolerant systems is to use N-modular hardware redundancy (many variations of the original scheme introduced by von Neumann in [1] exist): we perform the desired function in parallel by replicating the original system N times. The outputs of all replicas are compared, and the final result is chosen based on what the majority of them has agreed upon. Also, those in the minority are declared faulty.

Research in communications has extensively explored alternative, more efficient ways of utilizing redundancy for error detection and correction. Examples of such efficient schemes are the error correcting codes that are used when one transmits digital data through an imperfect channel, [2]: instead of replicating the links between two sites, or sending multiple replicas of a message (which amount to modular and time redundancy respectively), one sends only a fraction of additional or *parity* bits. The receiver uses the relations that govern the additional bits to try and correct possible bit-errors that took place during the transmission. In more complex systems that involve not only simple transmission of the data but also some form of processing on the data (e.g., computational or DSP systems), the application of such error correcting ideas is more challenging. Work in this direction includes *arithmetic* codes (see, for example, [3]) and algorithm-based fault tolerance (ABFT) techniques (introduced by Abraham, [4]–[6], and subsequently developed by others). These techniques have been quite successful, but they have to be cleverly tailored to the specific applications under consideration.

More broadly applicable and systematic approaches for introducing redundancy in general computational systems were studied recently by Beckmann, [7]–[9], and later by us, [10]–[12]. Beckmann's work focused on computa-

*This work has been supported in part by the Department of the Navy, Office of the Chief of Naval Research, contract number N00014-93-1-0686 as part of the Advanced Research Projects Agency's RASSP program.

†Address for correspondence: Room 36-615, MIT, Cambridge, MA 02139. Tel: (617) 253-0565 Fax: (617) 253-8495.

tions that can be modeled as abelian group operations[1], and used group homomorphisms both to introduce redundancy and to analyze its properties. Our work extended Beckmann's framework and analyzed operations that can be modeled as occurring in *semigroups* or *semirings*. Even though this is a very broad setting, we have been able to generalize most of Beckmann's results and to develop an algebraic framework for analyzing a large class of fault-tolerant computational systems.

This paper describes a mathematical framework for the design of fault-tolerant LTI systems in state form. Our approach, motivated by our earlier work, consists of mapping the original state vector into a higher dimensional space in a way that preserves the evolution and properties of the original system. This results in an embedding of the original system into a larger, redundant system. We are able to completely characterize all possible redundant systems of this type and to illustrate that our method essentially amounts to augmenting the original system with redundant *modes* that are *unreachable* but *observable* under fault-free conditions. Because these modes are not excited initially, they manifest themselves only when a fault takes place. We describe these results in Section 2 and present examples of error detection and correction in Section 3. We summarize our presentation and discuss future directions in Section 4.

2 Redundant LTI Systems

2.1 LTI State-Space Models

Linear time-invariant systems in state form constitute a well studied class of dynamic systems with a variety of applications, such as digital filter design, system simulation and model-based control, [13]–[15]. Throughout the analysis in this paper, we assume that the state, input and output vectors have elements drawn from $\mathbb{R}$, the field of real numbers. Although our discussion is focused on the discrete-time case, most of our results and examples can be translated to the continuous-time case in a straightforward manner.

An LTI system is represented in state form by the following pair of equations:

$$x[k+1] = Ax[k] + Bu[k] \quad (1)$$
$$y[k] = Cx[k] + Du[k] \quad (2)$$

where k is the discrete-time index, $x[k]$ is the *state vector*, $u[k]$ is the *input vector*, and $y[k]$ is the *output vector*. Assume that the vector x is n-dimensional (n is also known as the *system order*), u is p-dimensional and y is m-dimensional. Eq. (1) is referred to as the *state evolution* equation and eq. (2) is the *output* equation; $A, B, C,$ and D are constant matrices of appropriate dimensions. One can obtain equivalent state-space models (with n-dimensional state vector $x'[k]$) through similarity transformation, [14], [15]:

$$\begin{aligned} x'[k+1] &= (T^{-1}AT)x'[k] + (T^{-1}B)u[k] \\ &\equiv A'x'[k] + B'u[k] \\ y[k] &= (CT)x'[k] + Du[k] \\ &\equiv C'x'[k] + D'u[k] \end{aligned}$$

where T is an invertible $n \times n$ matrix such that $x[k] = Tx'[k]$. The initial conditions for the transformed system can be obtained as $x'[0] = T^{-1}x[0]$. Systems related in such a way are known as *similar* systems.

Given an input-output specification of an LTI system, there exist many possible ways of *realizing* it, that is, relating it to a particular state-space representation as in eqs. (1) and (2) above. A realization that uses the minimum possible number of state variables is called *minimal*. The analysis and design of LTI state-space systems has aimed almost exclusively at such systems. As we have already argued in the introduction, redundancy is not necessarily undesirable because it can be used to provide fault tolerance to a given system. In the next section we study ways of systematically introducing redundancy in order to achieve error detection and correction.

2.2 Systematic Introduction of Redundancy

In order to achieve error detection and correction in an LTI state-space system S of dimension n, we embed this system in a redundant state-space system $\mathcal{S}$ of dimension $\eta \equiv n + d$, $d > 0$. The state vector of $\mathcal{S}$ at the k-th time step, $\xi[k]$, provides complete information about $x[k]$, the state of the original system S at time k, but the d additional state variables of can be used for error protection. We develop this claim in more detail next. For the rest of this paper, we essentially ignore the output equation (2) and focus on the state evolution equation (1).

Let the state evolution equation of the original system S be given by $x[k+1] = Ax[k] + Bu[k]$, and the evolution of the redundant system $\mathcal{S}$ by

$$\xi[k+1] = \mathcal{A}\xi[k] + \mathcal{B}u[k] \ . \quad (3)$$

We wish to ensure that, at every time step k, the state vector $x[k]$ (and therefore the output vector $y[k]$) can be recovered from $\xi[k]$ through a *constant* $n \times \eta$ *decoding* matrix L, i.e.

$$x[k] = L\xi[k] \quad \text{for} \quad \text{all} \quad k \ .$$

(Note that under the assumptions so far the redundant system $\mathcal{S}$ can be regarded as a *cover* for S. The term "cover" has been used mostly in the language of finite automata, [16]: a finite automaton $\mathcal{S}$ is a cover for an automaton S

[1] These results can be generalized to systems whose operations take place in a ring or a field, because of the underlying group structure.

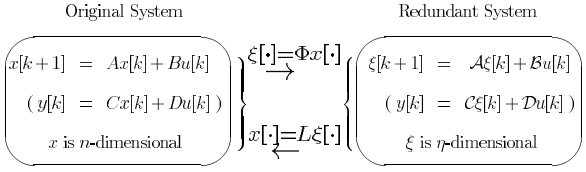

Figure 1: Relationships between original and redundant systems.

if, given the input of S, $\mathcal{S}$ is cable of concurrently simulating S, that is, there exists a mapping that maps the state of $\mathcal{S}$ at any given time to the corresponding state of S — for more details refer to [16].)

In order to achieve fault-tolerance, we impose a design requirement on the states of the redundant system $\mathcal{S}$: there should exist a constant linear mapping from the set of states of S to the set of states of $\mathcal{S}$. This is a natural constraint from the perspective of using the redundancy in $\mathcal{S}$ in some useful way (though, in general, the mapping need not be linear). This linear mapping can be represented by an $\eta \times n$ *encoding* matrix Φ which satisfies

$$\xi[k] = \Phi x[k] \quad \text{for} \quad \text{all} \quad k \ .$$

Under the above assumptions, $L\Phi = I_n$ (where I_n is the $n \times n$ identity matrix). Note that this equation by itself does not uniquely fix L or Φ. Fault detection is straightforward: since the redundant state vector $\xi[\cdot]$ must be in the column space of Φ under fault-free conditions, all we need to check is that at each time step k, $\xi[k]$ lies in the column space of Φ. Equivalently, we can check that $\xi[k]$ is in the null space of an appropriate *parity check matrix* Θ, so $\Theta\xi[k] = 0$ under fault-free conditions. We illustrate ways of obtaining the matrix Θ later in this section.

Figure 1 gives a picture of the relationships between the original and redundant systems. The dynamics of the system S on the left are governed by the matrices A, B, C, and D, whereas the dynamics of the redundant system $\mathcal{S}$ are governed by appropriately chosen matrices $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, $\mathcal{D}$. We can move between the two state vectors $x[\cdot]$ and $\xi[\cdot]$ using the encoding and decoding matrices (Φ and L respectively). We are now in a position to prove the main theorem of this section.

Theorem In the setting described above, a system $\mathcal{S}$ (of dimension $\eta \equiv n + d$, $d > 0$) is a redundant version of S if and only if it is *similar* to a *standard* redundant system $\mathcal{S}_\sigma$ whose state evolution equation is given by

$$\xi_\sigma[k+1] = \begin{bmatrix} A & A_{12} \\ 0 & A_{22} \end{bmatrix} \xi_\sigma[k] + \begin{bmatrix} B \\ 0 \end{bmatrix} u[k] \ . \quad (4)$$

Here A and B are the matrices in eq. (1), A_{22} is a $d \times d$ matrix that describes the redundant modes that have been added, and A_{12} is an $n \times d$ matrix that describes the coupling between the redundant and non-redundant modes. Associated with this standard redundant system is the standard decoding matrix $L_\sigma = \begin{bmatrix} I_n & 0 \end{bmatrix}$, the standard encoding matrix $\Phi_\sigma = \begin{bmatrix} I_n \\ 0 \end{bmatrix}$ and the standard parity check matrix $\Theta_\sigma = \begin{bmatrix} 0 & I_d \end{bmatrix}$.

Proof Let $\mathcal{S}$ be a redundant version of S. From $L\Phi = I_n$, L is a full-row-rank $n \times \eta$ matrix and there exists an invertible $\eta \times \eta$ matrix $\mathcal{T}_1$ such that $L\mathcal{T}_1 = \begin{bmatrix} I_n & 0 \end{bmatrix}$. If we apply the transformation $\xi[k] = \mathcal{T}_1\xi'[k]$ to the system $\mathcal{S}$, we obtain a similar system $\mathcal{S}'$ whose decoding mapping is $L' = L\mathcal{T}_1 = \begin{bmatrix} I_n & 0 \end{bmatrix}$, whereas the encoding mapping $\Phi' = \mathcal{T}_1^{-1}\Phi = \begin{bmatrix} I_n \\ K \end{bmatrix}$ (where K is a $d \times n$ matrix).

We can simplify things further by applying another transformation: $\xi'[k] = \mathcal{T}_2\xi''[k]$, where $\mathcal{T}_2 = \begin{bmatrix} I_n & 0 \\ K & I_d \end{bmatrix}$. We now obtain a redundant system $\mathcal{S}''$ (similar to both $\mathcal{S}$ and $\mathcal{S}'$) whose state evolution is given by

$$\begin{aligned} \xi''[k+1] &= (\mathcal{T}_2^{-1}\mathcal{T}_1^{-1}\mathcal{A}\mathcal{T}_1\mathcal{T}_2)\xi''[k] + (\mathcal{T}_2^{-1}\mathcal{T}_1^{-1}\mathcal{B})u[k] \\ &\equiv \mathcal{A}''\xi''[k] + \mathcal{B}''u[k] \ . \end{aligned} \quad (5)$$

By employing the above transformation, we have achieved a system for which L'' is still given by $\begin{bmatrix} I_n & 0 \end{bmatrix}$, whereas the encoding matrix is now $\Phi'' = \mathcal{T}_2^{-1}\Phi' = \begin{bmatrix} I_n \\ 0 \end{bmatrix}$.

Therefore, for all time steps k, and under fault-free conditions, $\xi''[k] = \Phi''x[k] = \begin{bmatrix} x[k] \\ 0 \end{bmatrix}$. Combining the state evolution equations of the original and redundant systems (eqs. (1) and (5) respectively), we see that

$$\begin{aligned} \xi''[k+1] &= \mathcal{A}''\xi''[k] + \mathcal{B}''u[k] \\ &\Rightarrow \\ \begin{bmatrix} Ax[k]+Bu[k] \\ 0 \end{bmatrix} &= \begin{bmatrix} \mathcal{A}''_{11} & \mathcal{A}''_{12} \\ \mathcal{A}''_{21} & \mathcal{A}''_{22} \end{bmatrix} \begin{bmatrix} x[k] \\ 0 \end{bmatrix} \\ &\quad + \begin{bmatrix} \mathcal{B}''_1 \\ \mathcal{B}''_2 \end{bmatrix} u[k] \ . \end{aligned}$$

We conclude that the following equations have to hold:

$$\begin{aligned} Ax[k] + Bu[k] &= \mathcal{A}''_{11}x[k] + \mathcal{B}''_1u[k] \\ 0 &= \mathcal{A}''_{21}x[k] + \mathcal{B}''_2u[k] \ . \end{aligned}$$

By setting the input $u[k] \equiv 0$ for all k, we see that $\mathcal{A}''_{11} = A$ and $\mathcal{A}''_{21} = 0$. With the input now allowed to be non-zero, we conclude that $\mathcal{B}''_1 = B$ and $\mathcal{B}''_2 = 0$.

The system $\mathcal{S}''$ is therefore in the form of the standard system $\mathcal{S}_\sigma$ in eq. (4) with appropriate decoding and encoding matrices. At this point, however, the check matrix Θ'' is given by

$$\Theta'' = \begin{bmatrix} 0 & P \end{bmatrix}$$

where P can be *any* invertible $d \times d$ matrix. A trivial third similarity transformation will ensure that the parity check matrix takes the from $\begin{bmatrix} 0 & I_d \end{bmatrix}$, while keeping the system in the standard form $\mathcal{S}_\sigma$ in eq. (4), except with

$A_{12} = \mathcal{A}''_{12}P$ and $A_{22} = P^{-1}\mathcal{A}''_{22}P$. The decoding, encoding and check matrices are then as claimed in the statement of the theorem.

The converse, namely that if $\mathcal{S}$ is similar to a standard $\mathcal{S}_\sigma$ as in (4), then it is a redundant version of (1), is easy to show. □

The above theorem establishes a complete characterization of all possible fault-tolerant designs (subject to our restrictions) of a given LTI state-space model. The additional modes introduced by the redundancy never get excited under fault-free conditions because they are initialized to 0 and they are unreachable from the input. Due to the existence of the coupling matrix A_{12}, the additional modes are not necessarily unobservable through the decoding matrix.

2.3 Error Model

A more detailed discussion of error detection and correction requires a particular error model. In this section we describe the sorts of hardware faults that might plausibly take place in the implementation of our systems, and the way we reflect these faults into our theoretical framework (i.e., we describe our error model).

There are two kinds of hardware faults: *transient* and *permanent* faults, [7]. A transient fault at time step k causes errors at that particular time step, but disappears at the following ones. Therefore, if the errors are corrected before the initiation of step $k+1$, the system will resume its normal mode of operation. A permanent fault, on the other hand, causes errors at all remaining time steps. Clearly, a permanent fault can be treated as a transient fault for each of the remaining time steps (assuming error correction at every single time step), but in certain cases one can deal with it in more efficient ways (e.g., reconfiguration). In the examples of Section 3 we will be mostly concerned with transient faults; in Section 3.3 we will deal with permanent faults.

The error model does not have to exactly mimic the actual fault mechanism. For example, we can model the error due to a fault in a multiplier as additive, or that of a fault in an adder as multiplicative[2]. However, efficient error models need to be close to reality; otherwise, a single actual fault might manifest itself as an unmanageable number of errors in the error model. Therefore, in order to evaluate the performance of our redundant system, we need to know its actual hardware implementation so that we can choose an efficient error model.

In most of the examples in Section 3 we assume that we implement our LTI systems using memory elements (delays), adders and multipliers (gains) that we interconnect in some appropriate way. These realizations can be represented using signal flow graphs, or delay-adder-gain diagrams as shown in Figure 2. Note that the same state-space description (matrices $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, and $\mathcal{D}$ for the redundant system) corresponds to a number of different delay-adder-gain diagrams and, consequently, it can have a number of different hardware realizations, [13]. This makes the connection with hardware failures more complicated. For example, in certain implementations a single fault in a multiplier or an adder can corrupt more than a single entry in the matrices $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, and $\mathcal{D}$ (and, consequently, more than one state variable). For the delay-adder-gain diagram in Figure 2, $\mathcal{A} = a$, $\mathcal{B} = 1$, $\mathcal{C} = b_1 + ab_0$, and $\mathcal{D} = b_0$. Clearly, a failure in a single multiplier can manifest itself in many ways: e.g., if the gain b_0 fails, $\mathcal{C}$ and $\mathcal{D}$ will be incorrect.

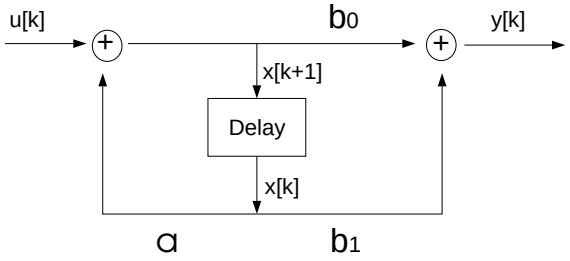

Figure 2: A delay-adder-gain diagram for a first-order LTI state-space system (single-input single-output case).

One way to avoid this problem is to assume that we implement our systems using delay-adder-gain diagrams in which the longest delay-free path is of length one. In such a case, the multiplier gains are directly reflected as the entries in the matrices of the state-space description, [13]. We can then model faults in the multipliers (and the adders) as corruptions in individual entries of the matrices $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, and $\mathcal{D}$. This is the assumption that we make in analyzing our examples in the next section[3].

The importance of the actual hardware implementation can also be seen from the following example: if our redundant system is directly implemented in the form (4), with the parity check matrix $\Theta_\sigma = \begin{bmatrix} 0 & I_d \end{bmatrix}$, then the redundancy is quite useless: under the assumptions of the previous paragraph, the only faults that are detected are the ones that have affected the redundant modes of the system at time step k (because the additional modes cannot be influenced by the original modes or the input). This is pointless, because our objective is to use the redundancy to protect the original system, not to protect redundancy itself. However, systems that are *similar* to the standard one can be designed to efficiently provide error protection.

[2] The faulty result r_f of a *real-number* multiplier can always be modeled in an additive error fashion as $r_f = r + e$ where r is the correct result and e is the additive error that has taken place. Similarly for an adder.

[3] A future step is to study more general descriptions by studying *factored state variable* descriptions, [13]. It is also possible to accommodate for implementations that are based on more general delay-adder-gain diagrams by looking at the techniques in Section 3.3, or by employing the computation trees in [17].

3 Examples of Fault-Tolerant Systems

In this section we present examples of achieving fault tolerance using the redundant systems developed in the previous section.

3.1 Triple Modular Redundancy

Triple modular redundancy (TMR) maintains three separate copies of the original system. These copies (modules) use *separate hardware* and operate identically under fault-free conditions. By comparing their state vectors at a given time step, one is able to detect transient or permanent errors. In fact, single errors can easily be corrected using a non-linear, but otherwise simple, voting scheme: we select the state agreed upon by two or more systems.

TMR in our LTI state-space setting corresponds to a system of the form

$$\xi[k+1] \equiv \begin{bmatrix} x^1[k+1] \\ x^2[k+1] \\ x^3[k+1] \end{bmatrix} = \begin{bmatrix} A & 0 & 0 \\ 0 & A & 0 \\ 0 & 0 & A \end{bmatrix} \xi[k] + \begin{bmatrix} B \\ B \\ B \end{bmatrix} u[k] \qquad (6)$$

where the initial conditions are chosen so that the state vectors $x^1[k]$, $x^2[k]$ and $x^3[k]$ of the three subsystems evolve in the same way as the original one (i.e., $x^1[0] = x^2[0] = x^3[0] = x[0]$). The encoding matrix Φ is given by $\begin{bmatrix} I_n \\ I_n \\ I_n \end{bmatrix}$, whereas the decoding mapping L can be $\begin{bmatrix} I_n & 0 & 0 \end{bmatrix}$, $\begin{bmatrix} 0 & I_n & 0 \end{bmatrix}$, $\begin{bmatrix} 0 & 0 & I_n \end{bmatrix}$, or others (e.g., convex combinations). In this example, we assume that $L = \begin{bmatrix} I_n & 0 & 0 \end{bmatrix}$. The parity check matrix Θ can be simply[4] $\begin{bmatrix} -I_n & I_n & 0 \\ -I_n & 0 & I_n \end{bmatrix}$. When a non-zero entry appears on the upper (respectively, lower) half of the $2n$-dimensional vector $\Theta\xi[k]$, we know that a fault took place in subsystem 2 (respectively, 3). When non-zero entries appear in both the top and bottom half-vectors, then a fault exists in subsystem 1.

The system is easily shown (for example with $\mathcal{T} = \begin{bmatrix} I_n & 0 & 0 \\ I_n & I_n & 0 \\ I_n & 0 & I_n \end{bmatrix}$, $\xi[k] = \mathcal{T}\xi_\sigma[k]$) to be similar to

$$\xi_\sigma[k+1] = \begin{bmatrix} A & 0 & 0 \\ 0 & A & 0 \\ 0 & 0 & A \end{bmatrix} \xi_\sigma[k] + \begin{bmatrix} B \\ 0 \\ 0 \end{bmatrix} u[k]$$

which is of the form described in the theorem of the previous section. The initial conditions are now $\xi_\sigma[0] = \mathcal{T}^{-1}\xi[0] = \begin{bmatrix} x[0] \\ 0 \\ 0 \end{bmatrix}$ where $x[0]$ is the initial condition associated with the original system. Note that all modes of the original system are replicated twice ($A_{22} = \begin{bmatrix} A & 0 \\ 0 & A \end{bmatrix}$) and there is no coupling ($A_{12} = 0$). The check matrix for the standard system is $\Theta_\sigma = \Theta\mathcal{T} = \begin{bmatrix} 0 & I_n & 0 \\ 0 & 0 & I_n \end{bmatrix}$.

Once the encoding matrix Φ is fixed, the additional freedom in choosing the decoding matrix L can be used to our advantage. For example, when our checking procedure detects *permanent* faults in the first subsystem, we can change our decoding matrix from $L = \begin{bmatrix} I_n & 0 & 0 \end{bmatrix}$ to $L = \begin{bmatrix} 0 & I_n & 0 \end{bmatrix}$. This will ensure that our overall system still outputs the correct result. In fact, this idea can be generalized a little bit. We discuss an adaptive framework with this flavor in Section 3.3.

3.2 Checksum and Linear Coding

Checksum

A scheme for detecting errors in LTI state-space systems was presented in [4] under the name "state variable filter". The basic idea was to include an extra state variable, $c[k]$, that forms the sum of all other state variables at each time step (i.e., $c[k] = \sum_{i=1}^{n} x_i[k]$). This was accomplished using the following redundant system:

$$\xi[k+1] \equiv \begin{bmatrix} x^1[k+1] \\ c[k+1] \end{bmatrix} = \begin{bmatrix} A & 0 \\ \sum_i a_i & 0 \end{bmatrix} \xi[k] + \begin{bmatrix} B \\ \sum_i b_i \end{bmatrix} u[k]$$

where a_i denotes the i-th row vector of the matrix A and b_i denotes the i-th row vector of matrix B. The encoding matrix Φ is now $\begin{bmatrix} I_n \\ 1 \quad 1 \ldots 1 \end{bmatrix}$, whereas the decoding matrix is $L = \begin{bmatrix} I_n & 0 \end{bmatrix}$. The initial condition is $\xi[0] = \begin{bmatrix} x[0] \\ \sum_{i=1}^{n} x_i[0] \end{bmatrix}$ so that under fault-free conditions

$$\xi[k] = \begin{bmatrix} x[k] \\ \sum_{i=1}^{n} x_i[k] \end{bmatrix} \quad \text{for all } k\,.$$

The parity check matrix is given by $\Theta = \begin{bmatrix} 1 & 1 & \ldots & 1 & -1 \end{bmatrix}$, i.e., it checks that the variable c is indeed the sum of all original state variables. When one of the state variables is updated incorrectly, then the parity check is not 0 and we are able to detect that a fault has taken place.

The above system is included in our framework. Indeed the redundant system is similar to the standard form

$$\xi_\sigma[k+1] = \begin{bmatrix} A & 0 \\ 0 & 0 \end{bmatrix} \xi_\sigma[k] + \begin{bmatrix} B \\ 0 \end{bmatrix} u[k]\,.$$

[4] Other parity check matrices are also possible.

The transformation matrix we need to use is $\mathcal{T} = \begin{bmatrix} I_n & 0 \\ 1 \; 1 \ldots 1 & 1 \end{bmatrix}$. The initial condition in the standard case is $\xi_\sigma[0] = \begin{bmatrix} x[0] \\ 0 \end{bmatrix}$; the parity check matrix of the standard system is $\Theta_\sigma = [\, 0 \quad 0 \ldots 1 \,]$ and simply checks that the $(n+1)$-st state variable is zero.

The importance of the actual hardware implementation and the need to assume an implementation that corresponds to a delay-adder-gain diagram with delay-free paths of unit length (or use the techniques pointed out in Section 2.3) can be seen from the following simple example. Suppose that our hardware calculates $c[k+1]$ by first calculating all $x_i^1[k+1]$ (which we have to calculate anyway) and then setting $c[k+1] = \sum_{i=1}^n x_i^1[k+1]$. This is a perfectly valid implementation. Its delay-adder-gain diagram, however, has delay-free paths of length greater than one, and an error in the calculation of state variable $x_i^1[k+1]$ will also appear in $c[k+1]$. In fact, our system will not be able to detect the error because the parity check will still be valid. It is crucial that $c[k+1]$ be calculated a different scheme, otherwise we run the risk of adding redundancy that checks itself.

Linear Codes

The checksum scheme above is a very basic one because it only provides single-error detection. Using our framework, we can develop schemes that provide detection *and correction* of *multiple* transient faults. The following is a simple motivating example to illustrate the idea. Let the original system be

$$x[k+1] = \begin{bmatrix} .2 & 0 & 0 & 0 \\ 0 & .5 & 0 & 0 \\ 0 & 0 & .1 & 0 \\ 0 & 0 & 0 & .6 \end{bmatrix} x[k] + \begin{bmatrix} 3 \\ -1 \\ 7 \\ 0 \end{bmatrix} u[k] \; .$$

This system can be protected against single transient errors in the state variables. First consider using three additional modes, implemented in the standard redundant form:

$$\xi_\sigma[k+1] = \begin{bmatrix} .2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & .5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & .1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & .6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & .2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & .5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & .3 \end{bmatrix} \xi_\sigma[k] + \begin{bmatrix} 3 & -1 & 7 & 0 & 0 & 0 & 0 \end{bmatrix}^T u[k] \; .$$

Error detection in this form requires checking that $\Theta_\sigma \xi_\sigma[k]$ is 0, where Θ_σ is the parity check matrix given by $\Theta_\sigma = [\, 0 \quad I_3 \,]$. However, as we argued in the previous section, redundant systems in standard form cannot be used for detecting or correcting errors in the *original* modes: given a faulty state vector $\xi_\sigma^f[k]$, the fact that $\Theta_\sigma \xi_\sigma^f[k] \neq 0$ will simply mean that an error took place in the calculation of the *redundant* modes. What we would really like is to protect against errors that appear in the original modes. One way to achieve this is to employ a system similar to the standard redundant system, but with the following parity check matrix:

$$\Theta = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} . \qquad (7)$$

The choice of this Θ is motivated by the structure of Hamming codes in communications, see [2]. With a suitable similarity transformation, the corresponding redundant system is

$$\xi[k+1] = \begin{bmatrix} .2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & .5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & .1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & .6 & 0 & 0 & 0 \\ 0 & -.3 & .1 & 0 & .2 & 0 & 0 \\ .3 & 0 & 0 & -.1 & 0 & .5 & 0 \\ .1 & 0 & .2 & -.3 & 0 & 0 & .3 \end{bmatrix} \xi[k] + \begin{bmatrix} 3 & -1 & 7 & 0 & -9 & -2 & -10 \end{bmatrix}^T u[k] \; .$$

Using this equivalent system, we can detect and locate transient faults that cause the value of a single state variable to be incorrect at a particular time step. To do this, we check for non-zero entries in the vector $\theta[k] \equiv \Theta \xi[k]$. If, for example, $\theta_1[k] \neq 0$, $\theta_2[k] \neq 0$, and $\theta_3[k] \neq 0$, then the value of $\xi_1[k]$ is corrupted; if $\theta_1[k] \neq 0$, $\theta_2[k] \neq 0$, and $\theta_3[k] = 0$, then a fault has corrupted $\xi_2[k]$; and so forth. Once the erroneous variable is located, we can easily correct it using any of the parity equations in which it appears. For example, if $\xi_2[k]$ is corrupted, we can calculate the correct value by setting $\xi_2[k] = -\xi_1[k] - \xi_3[k] - \xi_5[k]$ (i.e., using the first parity equation). Since the faults are transient, the operation of the system will resume normally in the following steps.

The above approach uses a parity check matrix that forms a Hamming code, as we noted. Such codes can perform single-error correction very efficiently: instead of replicating the whole system, we only need to add a few redundant modes. In fact, as long as $2^d - 1 > \eta$ (where $\eta \equiv n + d$ is the dimension of the redundant system), we can guarantee the existence of a similar system that achieves single-error correction.

In contrast to the binary coding scheme presented above, the authors of [17] have developed a *real coding* scheme. This scheme (also included in our framework) performs single-error correction using only two additional state variables, but needs more complicated error detection and correction mechanisms[5]. The additional modes in the fault-tolerant scheme in [17] are set to 0. Clearly, as

[5]It would be worthwhile to compare the numerical properties and limitations of the two schemes imposed by finite precision arithmetic.

the example of this section has demonstrated, this is not necessary; in fact, in the next section we make use of the additional non-zero modes to design a scheme that adapts to permanent faults during operation.

3.3 Adaptive Decoding

Let us examine the TMR example in eq. (6) a little closer. A permanent fault in any subsystem can be detected using the parity check matrix $\Theta = \begin{bmatrix} -I_n & I_n & 0 \\ -I_n & 0 & I_n \end{bmatrix}$. The corrupted state variable(s) can be corrected e.g., by simple majority voting or by using any of the valid parity equations. However, since the fault is permanent, we would like to be able to avoid the overhead of error correction at each time step. In the TMR case, this can be done in a straightforward way: for example, once a fault permanently corrupts the first subsystem (by corrupting entries in its A or B matrices), we can switch our decoding matrix from $L = \begin{bmatrix} I_n & 0 & 0 \end{bmatrix}$ to $L = \begin{bmatrix} 0 & I_n & 0 \end{bmatrix}$ (or $L = \begin{bmatrix} 0 & 0 & I_n \end{bmatrix}$ or others) and ignore the parity checks that involve variables in the first subsystem. This ensures that the output of the redundant system is still correct. We can continue to perform error detection, but have lost the ability to do error correction. We now formalize and generalize this idea.

Consider again the redundant system $\mathcal{S}$ whose state evolution equation is given by eq. (3). Under fault-free conditions, $x[k] = L\xi[k]$ and $\xi[k] = \Phi x[k]$ for all k. Suppose that we implement this system using a delay-adder-gain interconnection with delay-free paths of unit length. A permanent fault in a multiplier of the system manifests itself as a corrupted entry in the matrices $\mathcal{A}$ or $\mathcal{B}$: the i-th state variable $\xi_i[k]$ (and other $\xi_j[\cdot]$ at later steps) will be corrupted if some of the entries[6] $\mathcal{A}(i, l_1)$ and/or $\mathcal{B}(i, l_2)$ for some l_1 in $\{1, 2, ..., n\}$, and some l_2 in $\{1, 2, ..., p\}$ are corrupted right after time step $k - 1$. We assume that we can locate the faulty state variable through the use of some linear error correcting scheme as in the previous section. We are allowed to adjust the decoding matrix L to a new matrix L_a, but we do not have control over the entries in $\mathcal{A}$ and $\mathcal{B}$. We would like to know which entry corruptions can be tolerated, and how to choose L_a.

The answers are rather straightforward. First, however, we need to find out which state variables will be corrupted eventually. If at time step k_0 we detect a corruption at the i-th state variable, then we know exactly the end result: at time step $k_0 + 1$, state variable $\xi_i[k_0]$ will corrupt the state variables that depend on it (let M_{i_1} be the set of indices of these state variables — including i); at time step $k_0 + 2$, the state variables with indices in set M_{i_1} will corrupt the state variables that depend on them; let their indices be in set M_{i_2} (which includes M_{i_1}); and so on. Eventually, the final set of indices for all corrupted state variables is given by the finite set M_{i_f} (note that $M_{i_f} = M_{i_\eta} = M_{i_1} \cup M_{i_2} \cup M_{i_3} ... \cup M_{i_\eta}$). The sets of indices M_{i_f} for all i in $\{1, 2, ..., \eta\}$ can be calculated in an efficient manner by computing $R(\mathcal{A})$, the *reachability matrix* of $\mathcal{A}$, as outlined in [18].

Once we have detected a fault at the i-th state variable, we know that our new decoding matrix L_a (if it exists) should not make use of state variables with indices in M_{i_f}. Equivalently, we ask the question: does there exist a decoding matrix L_a such that $L_a\Phi_a = I_n$? Here, Φ_a is the same as the original encoding matrix Φ except that $\Phi_a(j, l)$ is set to zero for all j in M_{i_f}, with l in $\{1, 2, ..., n\}$. If Φ_a is full-column-rank, such an L_a exists. In this case, our redundant system can withstand corruption of entries in the i-th row(s) of $\mathcal{A}$ and/or $\mathcal{B}$; any L_a that satisfies $L_a\Phi_a = I_n$ is suitable.

Examples

TMR is clearly a special case of the above formulation: corruption of a state variable of the first subsystem is guaranteed to remain within the first subsystem. Therefore $M_f \subseteq \{1, 2, ..., n\}$ and (very conservatively)

$$\Phi_a = \begin{bmatrix} 0 \\ I_n \\ I_n \end{bmatrix} .$$

Two possible L_a's are (among others) $L_a = \begin{bmatrix} 0 & I_n & 0 \end{bmatrix}$ and $L_a = \begin{bmatrix} 0 & 0 & I_n \end{bmatrix}$.

Less obvious is the following case (based on the linear coding example of the previous section). Suppose

$$\xi[k+1] = \begin{bmatrix} .2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & .5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & .1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & .6 & 0 & 0 & 0 \\ 0 & -.3 & .1 & 0 & .2 & 0 & 0 \\ .3 & 0 & 0 & -.1 & 0 & .5 & 0 \\ .1 & 0 & .2 & -.3 & 0 & 0 & .3 \end{bmatrix} \xi[k] + \begin{bmatrix} 3 & -1 & 7 & 0 & -9 & -2 & -10 \end{bmatrix}^T u[k]$$

where $L = \begin{bmatrix} I_4 & 0 \end{bmatrix}$ and Φ is given by

$$\Phi = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -1 & -1 & -1 & 0 \\ -1 & -1 & 0 & -1 \\ -1 & 0 & -1 & -1 \end{bmatrix} .$$

If $\mathcal{A}(2, 2)$ (whose value is .5) becomes corrupted, then the set of indices of corrupted state variables is clearly

[6] We use $A(i, l)$ to denote the element in the i-th row and the l-th column of matrix A.

$M_{2_f} = \{2, 5\}$ and Φ_a is given by

$$\Phi_a = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ -1 & -1 & 0 & -1 \\ -1 & 0 & -1 & -1 \end{bmatrix}.$$

A suitable L_a is given by

$$L_a = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & -1 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}.$$

Using this L_a, the redundant system can continue to function properly (that is, provide the correct state vector $x[k]$ for all future k) despite the corrupted entry $\mathcal{A}(2, 2)$. We can still use the parity check matrix of eq. (7) for fault detection, except that the checks involving the second and/or fifth state variables (i.e., the first and second checks in $\Theta\xi[k]$) are invalid.

4 Conclusion

We have outlined a systematic procedure for introducing controlled redundancy into linear time-invariant systems in state form. Our approach maps the state vector of the original system into a larger, redundant space, while ensuring that the evolution in the redundant space will preserve the evolution and properties of the original system. The added redundancy, through proper hardware implementation, can be channeled towards achieving error detection and correction under hardware faults. We have demonstrated ways in which this can be achieved by several examples. Moreover, we have characterized all appropriate fault-tolerant designs for such systems. Our current work focuses on extending some of our results to redundant Petri net models, max-plus state-space systems, and other classes of dynamic systems in state form.

References

[1] J. von Neumann, *Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components*. Princeton University Press, 1956.

[2] S. B. Wicker, *Error Control Systems*. Englewood Cliffs, New Jersey: Prentice Hall, 1995.

[3] T. R. N. Rao, *Error Coding for Arithmetic Processors*. New York: Academic Press, 1974.

[4] K.-H. Huang and J. A. Abraham, "Algorithm-based fault tolerance for matrix operations," *IEEE Transactions on Computers*, vol. 33, pp. 518–528, June 1984.

[5] V. S. S. Nair and J. A. Abraham, "Real-number codes for fault-tolerant matrix operations on processor arrays," *IEEE Transactions on Computers*, vol. 39, pp. 426–435, April 1990.

[6] J.-Y. Jou and J. A. Abraham, "Fault-tolerant FFT networks," *IEEE Transactions on Computers*, vol. 37, pp. 548–561, May 1988.

[7] P. E. Beckmann, *Fault-Tolerant Computation Using Algebraic Homomorphisms*. PhD thesis, EECS Department, Massachusetts Institute of Technology, Cambridge, MA, 1992.

[8] P. E. Beckmann and B. R. Musicus, "Fast fault-tolerant digital convolution using a polynomial residue number system," *IEEE Transactions on Signal Processing*, vol. 41, pp. 2300–2313, July 1993.

[9] P. E. Beckmann and B. R. Musicus, "A group-theoretic framework for fault-tolerant computation," in *Inter. Conf. on Acoustics, Speech, and Signal Processing*, vol. V, pp. 557–560, 1992.

[10] C. N. Hadjicostis, "Fault-tolerant computation in semigroups and semirings," M. Eng. thesis, Massachusetts Institute of Technology, Cambridge, MA, 1995.

[11] C. N. Hadjicostis and G. C. Verghese, "Fault-tolerant computation in semigroups and semirings," in *Proceedings of Inter. Conf. on Digital Signal Processing*, vol. 2, (Limassol, Cyprus), pp. 779–784, 1995.

[12] C. N. Hadjicostis and G. C. Verghese, "Fault-tolerant computation in semigroups and semirings." Submitted for publication, 1996.

[13] R. A. Roberts and C. T. Mullis, *Digital Signal Processing*. Reading, Massachusetts: Addison-Wesley, 1987.

[14] D. G. Luenberger, *Introduction to Dynamic Systems: Theory, Models, & Applications*. New York: John Wiley & Sons, 1979.

[15] T. Kailath, *Linear Systems*. Englewood Cliffs, New Jersey: Prentice-Hall, 1980.

[16] A. Ginzburg, *Algebraic Theory of Automata*. New York: Academic Press, 1968.

[17] A. Chatterjee and M. d' Abreu, "The design of fault-tolerant linear digital state variable systems: theory and techniques," *IEEE Transactions on Computers*, vol. 42, pp. 794–808, 1993.

[18] J. P. Norton, "Structural zeros in the modal matrix and its inverse," *IEEE Transactions on Automatic Control*, vol. AC-25, pp. 980–981, 1980.