DISCRETE EVENT DEVELOPMENT FRAMEWORK FOR HIGHLY RELIABLE SENSOR FUSION SYSTEMS

MOHD. ROKONUZZAMAN


CENTRE FOR NFLD. STUDIES
MEMORIAL UNIVERSITY
OF NEWFOUNDLAND

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA

UMI®
800-521-0600

Memorial
University of Newfoundland

This is to authorize the Dean of Graduate Studies to deposit two copies of my thesis/~~report entitled~~

DISCRETE EVENT DEVELOPMENT FRAMEWORK FOR HIGHLY RELIABLE SENSOR FUSION SYSTEMS

in the University Library, on the following conditions. I understand that I may choose only ONE of the Options here listed, and may not afterwards apply for any additional restriction. I further understand that the University will not grant any restriction on the publication of thesis/~~report~~ abstracts.

(After reading the explanatory notes at the foot of this form, delete TWO of (a), (b) and (c), whichever are inapplicable.)

The conditions of deposit are:

(a) that two copies are to be made available to users at the discretion of their custodians,

OR

(b) that access to, and quotation from, this thesis/report is to be granted only with my written permission for a period of one year from the date on which the thesis/report, after the approval of the award of a degree, is entrusted to the care of the University, namely, _____ 19 ____, after which time the two copies are to be made available to users at the discretion of their custodians,

OR

(c) that access to, and quotation from, this thesis/report is to be granted only with my written permission for a period of _____ years from the date on which the thesis/report, after approval for the award of a degree, is entrusted to the care of the University; namely, _____, 19 ____; after which time two copies are to be made available to users at the discretion of their custodians.

Date 08-02-99 Signed _____

_____ Witnessed by _____

Dean of Graduate Studies

NOTES

1. *Restriction (b) will be granted on application, without reason given.*

   However, applications for restriction (c) must be accompanied with a detailed explanation, indicating why the restriction is thought to be necessary, and justifying the length of time requested. Restrictions required on the grounds that the thesis is being prepared for publication, or that patents are awaited, will not be permitted to exceed <u>three</u> years.

   Restriction (c) can be permitted only by a Committee entrusted by the University with the task of examining such applications, and will be granted only in exceptional circumstances.

2. *Thesis writers are reminded that, if they have been engaged in contractual research, they may have already agreed to restrict access to their thesis until the terms of the contract have been fulfilled.*

Discrete Event Development Framework for Highly Reliable Sensor Fusion Systems

By
©Mohd. Rokonuzzaman, B.Sc.Eng., M.Eng.

A THESIS SUBMITTED TO THE SCHOOL OF GRADUATE
STUDIES IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE
OF DOCTOR OF PHILOSOPHY

FACULTY OF ENGINEERING AND APPLIED SCIENCE
MEMORIAL UNIVERSITY OF NEWFOUNDLAND
APRIL, 1999

ST. JOHN'S NEWFOUNDLAND CANADA

Abstract

Intelligent Systems are being deployed increasingly in safety and mission critical applications. This thesis has synthesized a novel engineering methodology for developing highly reliable sensor fusion systems (SFS) of multi-sensori intelligent systems for the applications in the safety and mission critical environments. This methodology includes both the avoidance of faults during the development phase and the tolerance of sensor failures during the operation phase. Petri net based novel discrete event framework has been proposed to model SFS as discrete event dynamic system. This intuitive mathematical framework abstracts the SFS as a hierarchically finite state machine. The intuitive graphical nature of this framework has the potential to enhance the communication between the developer and the client to capture sensing requirements resulting in avoidance of requirement errors. The mathematical attribute enables the developer to analyze different attributes of the modeled SFS to ensure logical and temporal correctness of the performance of the system. This proposed discrete event framework has been verified by simulating the design of an example sensor fusion system. The reasoning basis of the architecture of the underlying computing system from this Petri net model of the SFS has also been developed to ensure the temporal correctness during the operation phase. The use of redundancy to tolerate failure of sensors has been experimentally verified. Overheads have been identified to incorporate hardware fault-tolerance in this proposed SFS framework to tolerate sensor faults during the operation phase. A novel scheme has been developed to manage these overheads in a predictable manner. A fault-tree based novel scheme has been proposed to measure the probability of failure of different levels of fusion due to the failure of different sensors. A computationally simple scheme to detect transients present on the sensor data stream has been proposed with extensive simulation results to enhance system performance in operation phase. The loss of time sensitive data during the fault clearance intervals compromises the effectiveness of fault-tolerance in the SFS. A parallel sensing based novel scheme has been proposed to restore sensor data lost during the fault clearance intervals. The effectiveness of this proposed scheme has been experimentally verified by restoring data lost during fault clearance intervals of a triple modular redundant optical sensor.

Acknowledgements

I would like to express my sincere gratitude to my thesis supervisor, Dr. Ray Gosine, for his active supervision of this research work. It is through his patience, understanding and advice that this work has been done. I would like to extend my gratitude to my thesis supervising committee members, Dr. John Quaicoe and Dr. Charles Randell, for their advice and guidance.

I am grateful to the members of my Ph.D. comprehensive examination committee, Dr. R. Venkatesan and Dr. Michael Hinchey, for their time and constructive advice. I am indebted to Dr. J. J. Sharp for his care, patience, advice and understanding.

This research work was supported by the NSERC/Canadian Space Agency Partnership Grant-"Sensor development and integration for autonomous experiments" with Petro-Canada Resources, Canpolar East Inc., and Atlantic Nuclear Services Ltd. I would like to express my special thanks to the participating organizations, McGill University, University of British Columbia, and C-CORE of Memorial University, of this research project. Particular thanks go to Dr. Ray Gosine, the Principal Investigator for this collaborative research project. My special thanks to the staff of C-CORE for providing me pleasant and friendly working environment.

I would like to express my thanks to the staff of the Faculty of Engineering & Applied Science and the School of Graduate Studies for the help extended to me during my graduate studies at MUN.

Finally, I thank to my dear family for their patience, encouragement and blessings when I have been thousands of miles away from them.

Contents

List of Figures

1 Introduction

2 The Discrete Event Requirements Model of the Sensor Fusion System

3 Discrete Event Specifications of the Sensor Fusion System

4 Architecture of the Embedded Computing System to Implement the DEVS model of the SFS

5 Hardware Fault-Tolerance of the Sensor Fusion System (SFS)

6 The Detection of Sensor Faults Using Local Statistics

7 Restoration of Lost Sensor's Data During Fault-clearance Intervals

B Verification of Discrete Event Requirements Model of SFS by Simulation

C Verification of Discrete Event Specifications Model of SFS by Simulation

D The Architecture of the Embedded Computing System to Implement the Example SFS

E Improvement of the Reliability and the Required Overhead for the Incorporation of Hardware Fault-Tolerance in the Example SFS

F Detection of Sensor Faults in Multisensory System by Simulation

G Performance of a Fault Tolerant Optical Sensor Using Triple Modular Redundancy

List of Tables

1 Introduction

2 The Discrete Event Requirements Model of the Sensor Fusion System

3 Discrete Event Specifications of the Sensor Fusion System

4 The Architecture of the Embedded Computing System to Implement the DEVS Model of the SFS

5 Hardware Fault-Tolerance of the Sensor Fusion System (SFS)

6 The Detection of Sensor Faults Using Local Statistics

7 Restoration of Lost Sensor's Data During Fault-clearance Intervals

A A Design Problem to Verify the Discrete Event Framework to Engineer a Reliable Sensor Fusion System

B Verification of Discrete Event Requirements Model of SFS by Simulation

C Verification of Discrete Event Specifications Model of SFS by Simulation

D The Architecture of the Embedded Computing System to Implement the Example SFS

E Improvement of the Reliability and the Required Overhead for the Incorporation of Hardware Fault-Tolerance in the Example SFS

F Detection of Sensor Faults in Multisensory System by Simulation

G Performance of a Fault Tolerant Optical Sensor Using Triple Modular Redundancy

Acronyms and Symbols:

The following lists of acronyms and symbols appear throughout the body of this document. The acronyms are defined here in alphabetical order, and the symbols are defined approximately in the order in which they appear in the text.

Acronyms	Definitions
ADCs	Analog to digital converters.
ADCU	Analog to digital conversion unit.
APs	Analog processors.
AS	Action sub-system.
ASPU	Analog signal processing unit.
DAT	Data acquisition time.
DES	Discrete event dynamic system.
DEVR	Discrete event requirements.
DEVS	Discrete event specifications.
DPs	Digital processors.
DSPU	Digital signal processing unit.
FT	Fault tolerance.
I/O	Input and output.
IS	Intelligent system.
MMs	Memory modules.
MTTF	Mean time to failure
RB	Recovery block.
RDA	Reasoning about the DSPU architecture .
RS	Reasoning sub-system.
SA	Sensor array.
SFS	Sensor fusion sub-system.
STAE	Service time of aperiodic events.

Acronyms	Definitions
TBPE	Time between two successive periodic events.
TCAD	Time for computation of the acquired data.
WT	Waiting time.

Symbols	Definitions
Chapter 1:	
Pr_p	Perception process.
Pr_r	Reasoning process.
Pr_a	Actuation process.
Te_p	Time window for perception.
Te_r	Time window for reasoning.
Te_a	Time window for actuation.
E	An event.
I	Identification of an event.
t	Time of occurrence of an event.
Te	Service time of an event.
Pr	The process to be executed to serve an event.
PE_i	The *ith* periodic event.
AE_i	The *ith* aperiodic event.
Wq	Waiting time in the queue.
A_s	The failure event of SFS.
A_r	The failure event of RS.
A_p	The failure event of AS.
$R_{sa(t)}$	Reliability of sensor array.
$R_{aspu(t)}$	Reliability of analog signal processing unit.
$R_{adcu(t)}$	Reliability of analog to digital conversion unit.
$R_{dspu(t)}$	Reliability of digital signal processing unit.

Symbols	Definitions
$A_S(t)$	Availability of sensor fusion system.
$A_{sa}(t)$	Availability of sensors array.
$A_{aspu}(t)$	Availability of the ASPU.
$A_{adcu}(t)$	Availability of the ADCU.
$A_{dspu}(t)$	Availability of DSPU.
$P[Z]$	The probability of occurrence of terminal event Z
Chapter 2:	
S_n	The *nth* sensor.
S	Set of sensors.
Pr_n	The *nth* process.
Pc	A place to hold conditions.
Pc_n	The *nth* place to hold conditions.
Ts_i	The sensing period of the *ith* sensor.
$min(\Delta Ts_i)$	Maximum allowable service period for *ith* periodic event of the *ith* sensor.
Φ_i	Sensing phase of the *ith* sensor.
Φ_i^l	Lower limit of sensing phase of the *ith* sensor.
Φ_i^u	Upper limit of sensing phase of the *ith* sensor.
T_i^l	The lower limit of the period of the *ith* sensor.
T_i^u	The upper limit of the period of the *ith* sensor.
Gp	The grain size of period.
gp	The grain size of phase.
V_i	The set of virtual sensors of the *ith* sensor.
Prc	The set of preconditions.
Poc	The set of post conditions.
$PE_i.Poc$	The post conditions of *ith* periodic event.
$AE_j.Prc$	The preconditions of the *jth* aperiodic event.

Symbols	Definitions
$AE_j.Poc$	The postconditions of the *jth* aperiodic event.
Lpc_i	The life-time of the *ith* condition.
μ	The state of the DEVR model.
$\mu`$	The new state of DEVR model.
Cs_i	The cycle number of the *ith* periodic event.
Tp_{ij}	The execution time of the *jth* path driven by the *ith* periodic event.
a_{ij}	The number of times the *jth* aperiodic event executes in path *i*.
Bp_i	Maximum busy period to serve the *ith* sensor.
$Bp(t)$	Total busy period during the operational time t.
$U(t)$	The degree of utilization of the operational period
Chapter 3:	
Te_0	The execution time of a component at level '0'.
$T_{n.i}$	The execution time of the *ith* system at the *nth* level
Se	Sequential trace
Sc	Compound trace
c_n	The *nth* computing component
$\boldsymbol{C}$	Set of computing components
$\boldsymbol{Se}$	The trace vector.
$\boldsymbol{P}$	The position vector.
$\boldsymbol{Z}$	The set of all integer numbers.
P_{ij}	Position operator.
st	Pick one trace from a set of traces.
po	Runs more than one traces in parallel.
br	Select one trace from a set of traces to branch.
Sge_i	An elementary segment.

Sgc_i	A compound segment.
St_i	The execution time of the *ith* trace
Δt_i	The incremental change of execution time of *ith* computing component.
ΔT_i	The total change of execution.
C_i	The cost related to the per unit change of computation time for the *ith* component.
Sen_i	The sensitivity of the execution time of the *ith* computing component.
$Csen_i$	The cost sensitivity of the *ith* component.
Chapter 4:	
CC	The computational complexity vector.
CC.I	The computational load for integer.
CC.F	The computational load for floating point.
CC.D	The computational load for control flow.
CC.M	The computational load for memory operations.
CC.H	The system management overhead.
λ	The traffic arrival rate in the queue.
T_{eff}	The effective access time of a data unit.
Chapter 5:	
p^n	The probability of failure of nth parallel component.
$p(t)$	The physical signal.
$s(t)$	The operating state of the sensor.
$g(t)$	The electrical signal generated from the sensor.
M	The message space.
D	The decision space
$d(z)$	The decision rule
$f(t)$	The signal from the analog signal processor.

Symbols	Definitions
$ap(t)$	The operating state of the analog signal processor.
Chapter 6:	
$g(nT)$	Discrete sensor signal.
$\overline{g(K)}$	The mean of the *kth* segment.
$g^2(K)$	The variance of the *kth* segment.
Chapter 7:	
$s[n]$	The segment of signal acquired in a session.
$s_1[n]$	The segment of signal acquired by channel 1.
$s_1[n]$	The segment of signal acquired by channel 2.
F_i	The *ith* fault clearance interval.
$t_{ij}(k)$	The position of samples in the *ith* channel in the *jth* fault clearance interval.
$l_{ir}[u]$	The positions of undefined samples in the *ith* channel in the *rth* region.
$\tilde{s}[n]$	The recovered signal.
t_C	The comparison time taken by the comparator.
t_S	The selection time taken by the voter.
t_m	The switching time taken by the multiplexer.

Chapter 1

Introduction

1.1 The Overview of Intelligent Systems

An intelligent system (IS) perceives, reasons, and acts through the dynamic interaction of a set of discrete events within the specified time windows. A Petri net [1] based closed loop system model as shown in the Fig.1.1 represents this scenario of dynamic interaction. The processes Pr_p, Pr_r, and Pr_a corresponding to perception, reasoning, and action must be executed within the time windows Te_p, Te_r, and Te_a respectively. This type of system is being used increasingly in safety and mission critical operations in space, medicine, manufacturing, mining, undersea, and harsh environments. Some of these operations require unsupervised, autonomous functions. High reliability and fail safe characteristics are critical operational requirements of these systems.

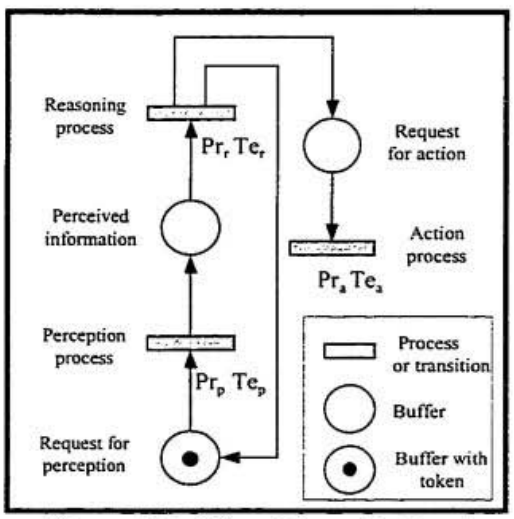


Figure 1.1: Closed loop Petri net model of an Intelligent System.

To achieve these requirements, faults must be avoided during both the development and operation phases of product life cycle. During the development phase, through the practice of appropriate formal methods, it is possible partially to realize this objective. To achieve these objectives in the operation phase, the system must have the ability to detect the failure of the constituting components and, if possible, to replace the failed component with a fault-free one. If there is no spare component for replacement, the

system should take necessary steps to avoid malfunctioning. The incorporation of these attributes in the different phases of the product life cycle will result in a highly reliable IS.

Perception by an IS is partially accomplished through fusing information from a set of complementary and/or redundant sensors [2]. This fusion of sensor data is performed in the Sensor Fusion Sub-system (SFS). The SFS acquires data from different sensors, fuses them to extract necessary information and sends them to the Reasoning Sub-system (RS). The RS sends appropriate commands to the Action Sub-system (AS) [1]. The data flow among different sub-systems in an intelligent system is shown in Fig. 1.2. The SFS is the subject of this present work, which addresses the system engineering aspect of the development of a highly reliable sensor fusion system. It is expected that the results of this work will equip the developers with necessary quantitative reasoning tools to develop reliable SFS. This scientific knowledge to engineer reliable SFS will partially realize the broader requirements of engineering highly reliable intelligent systems for safety and mission critical operations.

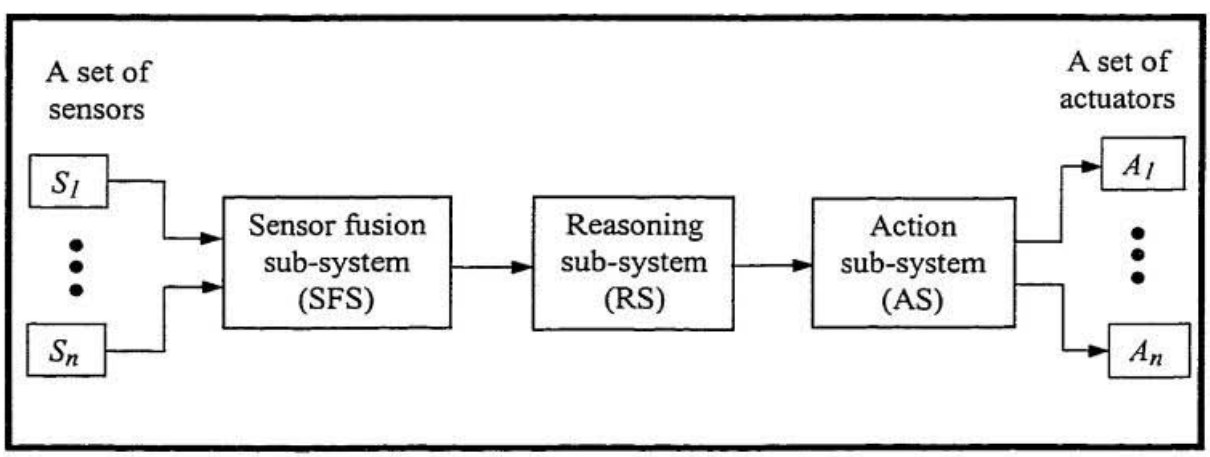


Figure 1.2 The block diagram representation of an intelligent system to show data flow (without feedback signals) among different sub-systems.

1.1.1 The Overview of Sensor Fusion

Sensor fusion techniques integrate information from multiple sensors in order to make an inference about a physical event, activity, or situation as shown in Fig. 1.3. The basic objective of multi-sensori data fusion is to achieve improved accuracies and more specific information than could be achieved by the use of a single sensor alone [2,3]. This refers to the synergistic use of the information provided by multiple sensory devices to assist in the accomplishment of a task by a system. The data integration from senses - sights, sounds, smells, tastes, and touch - by the ongoing cognitive process in the human's body is a common example of sensor fusion [4]. The timeliness, accuracy, and precision are salient attributes of such fusion process. A typical sensor fusion process consists of four activities: acquisition, processing, integration, and analysis as shown in Fig. 1.4. In the process of multi-sensori data integration, sensors can provide temporally related competitive, complementary, and independent information [6].

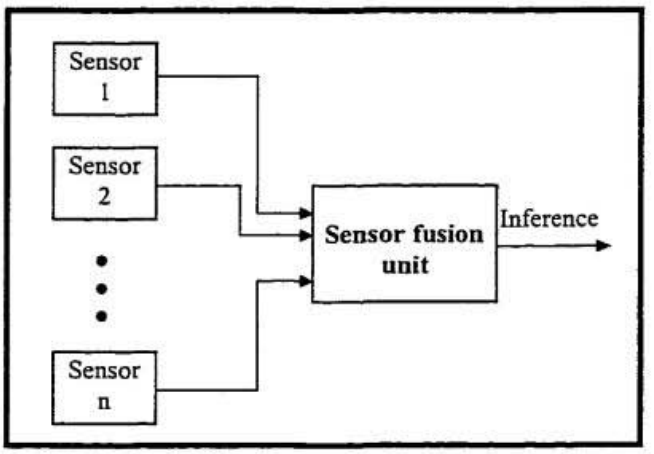


Figure 1.3: A block diagram model of a sensor fusion system.

From the perspective of input/output (I/O) characteristics, sensor fusion has been described in a three-level hierarchy: data, feature, and decision. This three-level hierarchical fusion is performed in five fusion processes as shown in Fig. 1.5 [7]. The hierarchical fusion of data from eight data sources is shown in Fig. 1.6 [6]. In this example system, data D_1 and D_2 are combined in the data integration step into feature F_{12}. In the similar way, D_3 and D_4, D_5 and D_6, and D_7 and D_8 are integrated to produce features F_{34}, F_{56}, and F_{78} respectively. In the next step, the features F_{12} and F_{34} are integrated into decision De_{1-4}. The integration of features F_{56} and F_{78} produces the decision De_{5-8}. In the

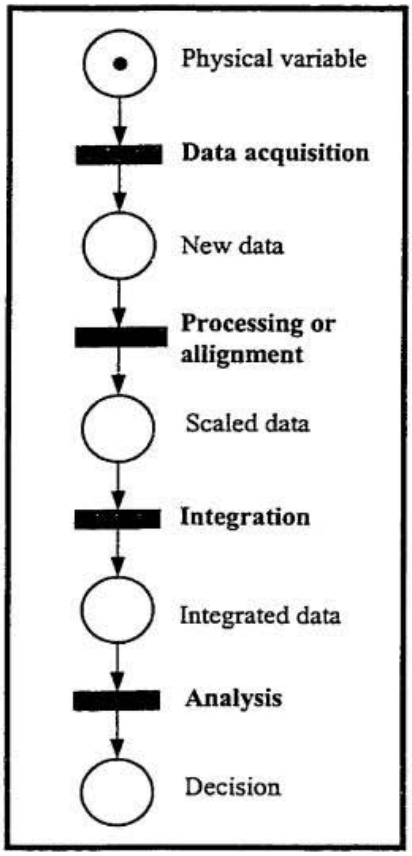


Figure 1.4: Data processing activities in atypical sensor fusion system.

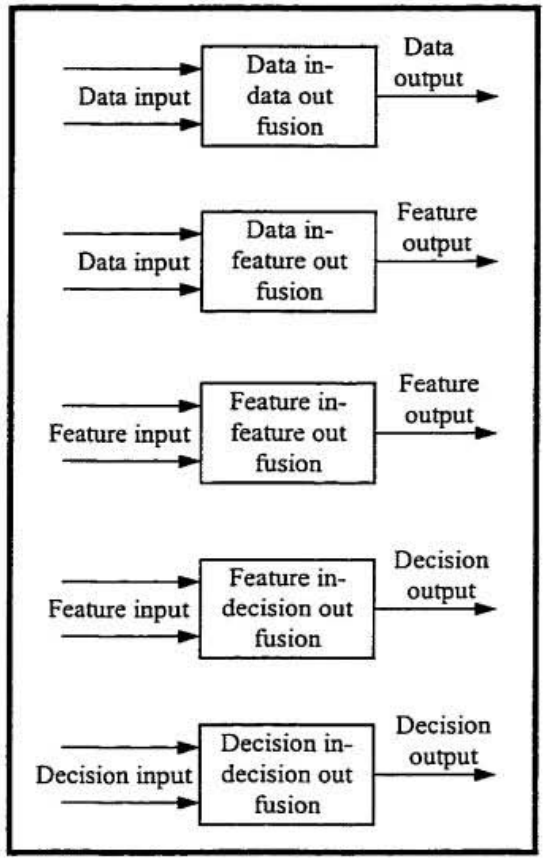


Figure 1.5: The fusion process from the perspective of input/output characteristics [7].

final stage, the local decisions are combined to produce the final decision about the sensing environment as $De_{1\text{-}8}$. The number of data integration steps is a function of the requirements of a particular sensing task. The example shown here is a generic representation of different data integration steps.

The sensor fusion process can be defined as the reunification of fragmented information in order to represent the information originally present in the environment. The fragmentation occurs due to inescapable fission that takes place during sensing due to the physical constraints of sensors (e.g., resolution, spatial coverage). In essence, this reunification of information is the main objective of sensor fusion and an ideal sensor

fusion system will be able to restore all information of interest in the environment from the data sensed by the multi-sensori suite [7].

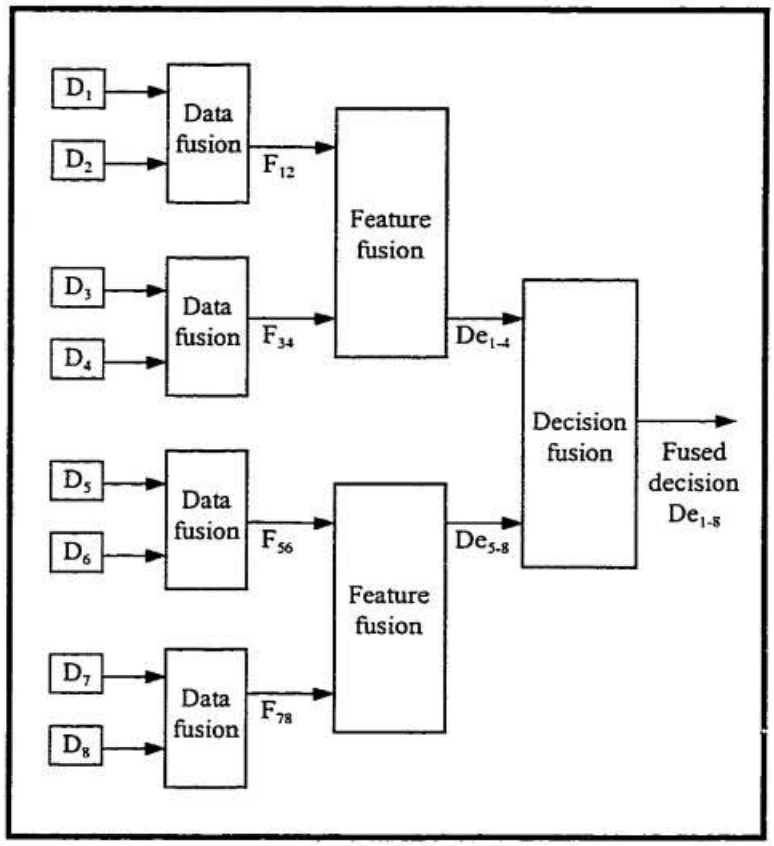


Figure 1.6: A hierarchical representation of data integration steps [6].

The problem environments requiring the applications of multi-sensori systems generate a large volume of data with differing spatial and temporal resolution, and often corrupted by noise and clutter. It is a formidable challenge for an engineer to design and develop a sensor fusion system to integrate data from multiple sensors in such environments, especially given the real-time constraints that are often imposed by the real-world needs [5]. The applications of sensor fusion are widespread; some typical application areas discussed in the literature are summarized in Table 1.1 [2].

The goals of sensor fusion are different for different application environments. The fusion objectives of a specific application typically include one or more of the following functions:

- Detection of the presence of an object or environmental condition.
- Identification of an object or event.
- Classification of detected objects or events.
- Tracking of an object or continued monitoring of an event.

Algorithms to fuse data from different sensors use techniques from several disciplines: signal processing, statistics, artificial intelligence, pattern recognition, cognitive psychology, and information theory. The rapid evolution of computer hardware technology (e.g., microprocessors and memory), advanced sensors and new techniques have led to new capabilities to combine data from multiple sensors for improved inferences. Implementation of such systems requires an understanding of basic terminology, data fusion processing models, and architectures. This work focuses on generic architectural aspect of SFS from the system engineering point of view.

Table 1.1: The potential applications of multi-sensori systems reproduced from [2].

Specific Applications	Inference sought by SFS	Primary observable data	Spatial coverage	Sensor platform
Robotics	Location, identification of obstacle, and objects to be manipulated.	• Optical signals • Acoustic signals • EM radiation • X-rays	Microscopic to tens of feet about the robot.	• Robot body
Medical diagnostics	Location, identification of tumors, abnormalities, and disease.	• X-rays • Acoustic signals • Optical signals • MRI • Chemical data	Human body	• Laboratory
Environmental monitoring	Identification, location of natural and manmade phenomena.	• SAR, Optical • Seismic • EM radiation • Chemical	Hundreds of miles	• Satellites • Aircraft • Ground-based • Underground
Preventive maintenance	Detection and characterization of system incipient faults	• Optical signals • EM radiation • Acoustic, vibration • Electric, magnetic • X-rays	Microscopic inspection to hundreds of feet	• Ships, • Aircraft • Ground-based systems (e.g.,factory equipments)
Ocean surveillance	Detection, tracking, identification of vessels, offshore structures, biological & chemical constituents icebergs, sea ice and fish stocks.	• SAR • Optical signals • EM radiation • Acoustic signals	Hundreds of nautical miles Air/surface/sub-surface	• Satellites • Ships, Aircraft • Submarines • Ground-based
Strategic warning and defense	Detection of indications of impending strategic actions. Detection and tracking of missiles, aircraft, ground-based targets.	• SAR • Optical signals • EM radiation's • Acoustic signals	• Hundreds of miles to global (strategic). • Miles (Tactical)	• Satellites • Aircrafts • Ships • Ground-based
Manufacturing	Measurements and inspections.	• Optical signals • Acoustic signals • Laser	Micrometers to few feet	Factory floor.

1.2 The Sensor Fusion Sub-System (SFS)

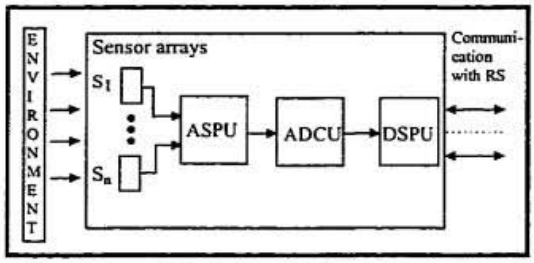


Figure 1.7: High level block diagram of the sensor fusion sub-system

An SFS can be developed either as a cooperating multi-node based system or a single node based system (centralized sensing system of mobile robots). In this thesis, the focus is on the engineering design methodology of single node based SFS. The unique issues related to the design of multi-node based sensor fusion systems are beyond the scope of this thesis. The block diagram of a single node based SFS is shown in Fig.1.7. The SFS consists of four major hardware components: sensor arrays (SA), analog signal processing unit (ASPU), Analog to digital conversion unit (ADCU), and digital signal processing unit (DSPU).

It should be stated that SFS could be abstracted as a finite state machine; the occurrence of an event (e.g., the detection of change of the environment) makes state transition of the SFS. To show real time behavior, the system must serve the events by executing appropriate processes within specified time windows. An event, E, can be defined as four-tuple vector,

$$E = \{I, t, Te, P_r\} \tag{1.1}$$

Here, I stands for identification, t for the time of occurrence, Te for the event service time, and Pr for the corresponding process. From temporal point of view, events are of two types: periodic and aperiodic events.

Periodic events are generated at regular time intervals and time is the forcing factor for their occurrence. For example, the periodic checking of the status of a process parameter (e.g., temperature, pressure) can generate events PE_i to run specific process Pr_i at time

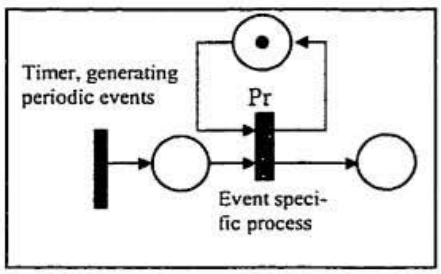


Figure 1.8: Petri net model of periodic event generation.

period Δt_i and can be represented by the equation

$$PE_i = \{I_i, t_0 + \sum_{j=0}^{n} \Delta t_i , Te_i, Pr_i\}; t_0, \textit{starting time} \quad (1.2)$$

A Petri net model of periodic event generation is shown as in Fig. 1.8. In this periodic event generation scheme, it has been considered that event generation period is larger than the event execution time.

Aperiodic events generated with the fulfillment of certain conditions fall into this category. These conditions depend upon the dynamic behavior of the environment. Failure of a component of the SFS (e.g, sensor) also generates aperiodic event. A Petri net model of aperiodic event generation is shown by Fig.1.9. In this model, process Pr, an aperiodic event specific process, will be executed when all of the input conditions, Pc_1,..,Pc_n, are satisfied. The fulfillment of the *ith* condition (e.g., Pc_5) will be represented by placing a token in the *ith* place. Therefore, the occurrences of these events are aperiodic in nature and can be represented by the following equation

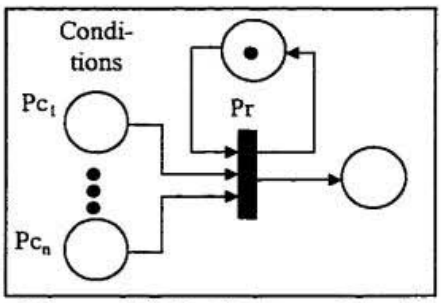


Figure 1.9: Petri net model of aperiodic event generation.

$$AE=\{I, t, Te, Pr\} \quad (1.3)$$

The interaction of these events inside the SFS results in a discrete event dynamic system (DES). These are real-time systems. The reliable operation of these systems requires that their functions maintain logical and temporal correctness. This work covers the following aspects of the development of the SFS:

1. **Discrete event requirements (DEVR)**
2. **Discrete event specifications (DEVS)**
3. **Reasoning about the DSPU architecture (RDA)**
4. **Fault-tolerance (FT)**

1.2.1 Importance of Discrete Event Requirements

The development of a SFS starts with the generation of the requirements document. This is also the time at which the most costly errors are introduced in terms of being the last and most difficult to find [8]. The requirements document, which corresponds to the behavioral specification of the system's activities, describes the system's discrete states of operation and the events that cause the system to change states [9]. This must reflect the required properties of the controlled physical process. To ensure high reliability, these requirements must be explicit and form the basis for the design. Therefore, the developer must be provided with mathematical tools to record such system properties [10]. These tools enable verification of the correctness and completeness of the requirements documents. Moreover, these mathematical tools should be natural, simple, and intuitive, so that the developer can use them as a communication media with the domain expert (i.e., the client).

The job of the SFS is to serve the dynamic interaction of a set of discrete events (both periodic and aperiodic) to satisfy the client's sensing requirements. Therefore, the requirements document of the SFS can be modeled as a DES. Petri nets are simple, natural, graphical, and mathematical tools, which can be used to model this requirement document as DES. Petri net models can be analyzed to verify the correctness and completeness of the modeled phenomena [1]. Petri nets, as a graphical tool, provide a powerful communication medium between the developer, typically requirement engineers, and the client. Due to the dynamic nature of Petri nets, SFS models can be treated as a virtual machine. The analysis of these models will help develop better insights into the client's sensing requirements. This formalism in the early stage of the development will help capture the system requirements more correctly. The subsequent development phases of the SFS will use this DEVR as a reference.

1.2.2 Importance of Discrete Event Specifications (DEVS)

The DEVR, the virtual dynamic machine, describes the system specifications from the user's perspective [9]. The SFS serves the RS through the service of discrete events. In this control paradigm, the RS expects to receive sensor responses to its requests within definite time windows. Each event is served through the dynamic interaction of a set of computing components known as processes as shown in Fig.1.10. The requirements document defines the time windows for the service of the set of events, $E_n=\{1,2,3,...,n\}$, where n is a positive integer. Therefore, the relationship of $E_n.Te$ with the allowable execution times of the components can be defined by the following equations:

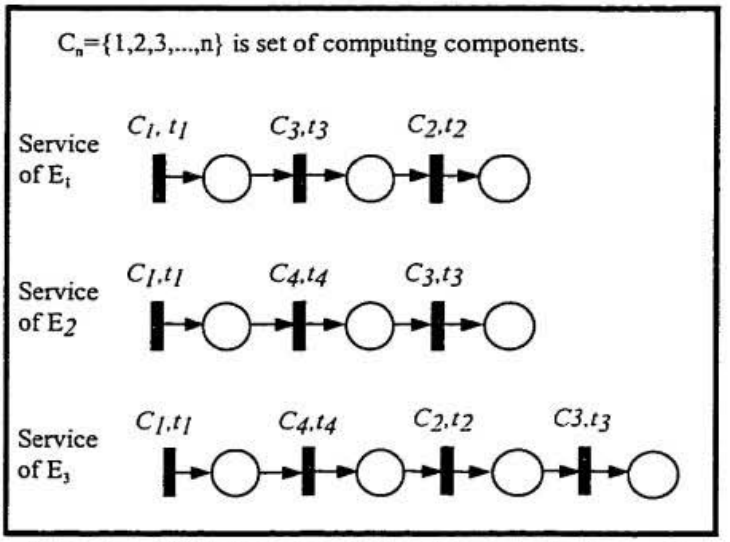


Figure 1.10: Petri net models of services of events through the dynamic interaction among serial computing components.

$$E_1.Te \leq (t_1+t_3+t_2) \quad (1.4)$$

$$E_2.Te \leq (t_1+t_4+t_3) \quad (1.5)$$

$$E_3.Te \leq (t_1+t_4+t_2+t_3) \quad (1.6)$$

Usually sensor fusion algorithms have different levels of computational complexity. The selection of a particular level of complexities of those computing components (sensor fusion algorithms) may satisfy one of those equations, but may not satisfy others. Therefore, there is a need to develop optimum algorithms within the DEVS formalism to decompose the event service time (i.e., corresponding process service time) into the constituent component execution times.

Using this model, simulation may be used to configure the computing components so that the SFS satisfies the DEVS model of the sensor fusion system. This event level specification will form the basis for the different phases of system development including design, fabrication, integration, testing, and updating. A guideline of the optimum component level research can be derived from this DEVS model.

1.2.3 Importance of Reasoning about the DSPU Architecture

The DEVS model of the SFS will be realized through the sequence of interactions among computing components. The operational scenario can be abstracted as a queuing system shown in Fig. 1.11. Now to serve an event properly, the following condition should be satisfied

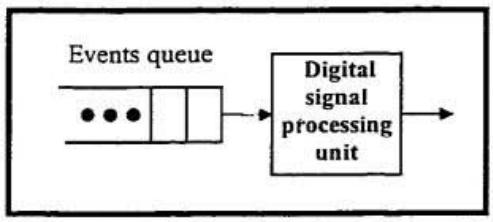


Figure 1.11: A model of services of events by the DSPU.

$$Wq + Ts \leq Te \tag{1.7}$$

Here, Wq is the expected waiting time of an event in the queue; Ts is the execution time of the corresponding process; Te is the event service time. Some of these events are periodic and some of them are aperiodic. The waiting time is a function of arrival rate. To handle this operation, it is necessary to have the DSPU with the following questions answered:

- **Computing nodes:**
  - Number
  - Specification of each node
  - Inter-node communication architecture

- **Stability in response time**

  Due to the random execution time delay of one or more components to serve random events (e.g., events generated due to failure of components), the event service time should not increase cumulatively.

1.2.4 Importance of Fault-Tolerance

Applications in safety and mission critical operations require highly reliable intelligent systems, which are fault-tolerant. In this operational scenario, the quality of data and the effective utilization of time are very critical factors. The following features can satisfy these objectives:

- high reliability (high probability of continuous proper function),
- high availability (relatively low down time associated with repairs),
- minimum time to recover from a detected fault,
- extremely low failure rates for short time periods,
- extremely high probability of transition to a safe state after occurrence of a malfunction,
- easy and timely on-line diagnosis and repair of faults.

Faults in both hardware and software contribute to system failure. Therefore, in order to develop a reliable SFS, both hardware and software fault-tolerance must be addressed.

1.2.4.1 Hardware Fault-tolerance

The fault tree in Fig.1.12 illustrates the impact of fault-tolerance of the SFS on system reliability. Here A_S, A_r, and A_p are the terminal events of the SFS, RS, and AS respectively. From a qualitative analysis of the fault tree, it is evident that failure of any sub-system (e.g. failure of SFS, $A_S=0$) will result in system failure since terminal events (sub-system failure) are connected by an 'AND' gate to the failure of the system.

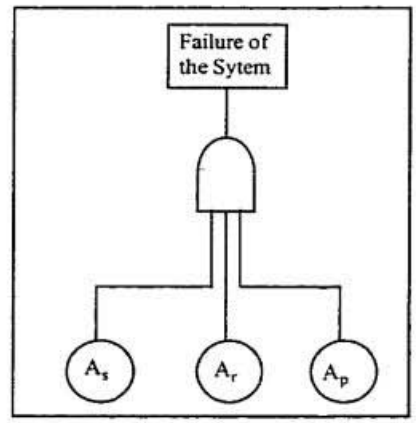


Figure 1.12: High level fault tree of an intelligent system.

1.2.4.1.1 The Reliability and Availability of the SFS

In this preliminary model of the SFS there are no redundant components. Under this condition, the reliability model of the SFS is illustrated in Fig.1.13. Here, $R_{sa}(t)$, $R_{aspu}(t)$, $R_{adcu}(t)$ and $R_{dspu}(t)$ represent the reliabilities of SA, ASPU, ADCU and DSPU respectively. The overall reliability of the SFS is given by Eq.(1.8).

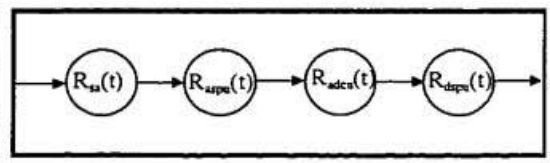


Figure 1.13: The reliability model of the sensor fusion sub-system

$$R_S(t) = R_{sa}(t).R_{aspu}(t).R_{adcu}(t).R_{dspu}(t) \tag{1.8}$$

The availability, $A_S(t)$, of the SFS is calculated using Eq.(1.9).

$$A_S(t) = A_{sa}(t).A_{aspu}(t).A_{adcu}(t).A_{dspu}(t) \tag{1.9}$$

Here, $A_S(t)$, $A_{Sa}(t)$, $A_{aspu}(t)$ $A_{adcu}(t)$, and $A_{dspu}(t)$ represent the availabilities of the SFS, the sensor array, the analog signal processing unit, the analog to digital conversion unit, and the digital signal processing unit respectively.

1.2.4.1.2 The Fault Tree of the SFS

The fault tree of the SFS depicts how component-level failures propagate through the system to cause a system-level failure (system-level undesired events). The component-level failures are called the terminal events. In this work, failures of SA, ASPU, ADCU and DSPU are considered terminal events. The fault tree of the SFS is shown in Fig. 1.14. Here, A_{sa}, A_{aspu}, A_{adcu} and A_{dspu} represent terminal events of the SA, ASPU, ADCU and DSPU respectively. The terminal event, Z, represents the failure of the SFS.

1.2.4.1.3 Quantitative Fault Tree Analysis

In this reliability analysis of the SFS, the SFS is considered to be a nonrepairable system. In this system, as all the events are statistically independent, the probability of Z at time t is given by the equation

$$P[Z] = P[A_{sa}]P[A_{aspu}]P[A_{adcu}]P[A_{dspu}] \tag{1.10}$$

Here, P[A_{sa}], P[A_{aspu}], P[A_{adcu}], and P[A_{dspu}] are the probabilities of A_{sa}, A_{aspu}, A_{adcu}, and A_{dspu} respectively at time t.

This analysis of the fault tree reveals that the failure of each component (probabilities of these components need not to be equal) of the SFS is equally responsible for the failure of the system. Therefore, there is a need to enhance the reliability of every unit to develop a reliable sensor fusion system.

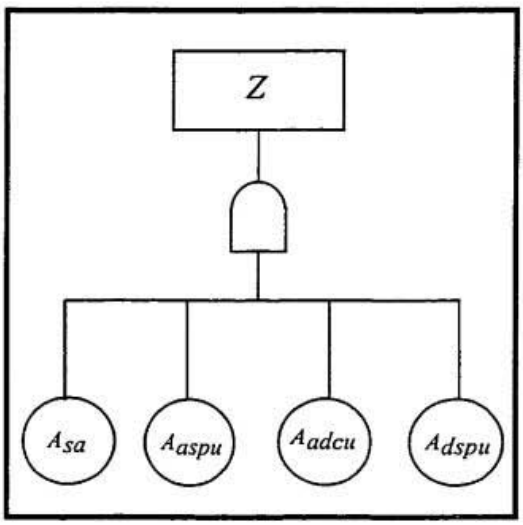


Figure 1.14: Fault tree of the sensor fusion sub-system.

1.2.4.2 Software Fault-tolerance

The requirement of high reliability of the SFS can be dealt with in two fundamental ways: *fault avoidance* and *fault tolerance* [11]. The different fault tolerance techniques are based on the premise that a complex system, no matter how carefully designed and validated, will encounter unpreventable operational faults and will contain residual design faults [11]. Due to success in hardware fault-tolerance using redundancy, some researchers have proposed the use of similar approaches to address this problem (i.e., software fault tolerance) [12]. These are well known recovery block (RB) and N-version programming approaches. It has been reported that these approaches are capable of increasing the reliability of the system [13]; but, it has also been argued that it is certainly not the case that when a fault appears, the system dynamically generates new corrected code [14]. A detailed study of fault tolerance indicated that the differences between software and hardware severely limit the application of hardware fault tolerance techniques to software [14]. This study also indicates that the current software fault tolerance techniques can be described as delayed debugging [14]. Most of the techniques used to achieve hardware fault tolerance enable systems to tolerate physical rather than design faults. The software is error prone due to design faults, certainly not due to aging of software components (i.e., code). There is no evidence that the level of reliability required in the safety critical software can be achieved using redundancy or N-version programming approaches [15]. Therefore, this thesis does not address the problem of software reliability emulating the concept of hardware fault tolerance or using N-version programming approach. It has been reported that the failure to use the system level approach to develop software systems for safety and mission critical operations appears to be the main problem in achieving the required level of reliability [15]. Therefore, it is believed that the use of system level approach based on DES formalism in the different phases of development of the SFS will help the designer to realize the required level of reliability for safety and mission critical operations. Moreover, this is beyond the scope of this thesis to address the software fault-tolerance aspect in a comprehensive manner.

1.3 Literature Review

The architecture of sensor fusion system in the block diagram level as shown in Fig.1.15(a) and Fig 1.15(b) has been reported [2]. Due to the lack of a mathematical formalism, this architecture cannot be simulated to verify logical and temporal correctness. Moreover, this architecture does not provide the framework for different modes of data integration in the same sensor fusion system.

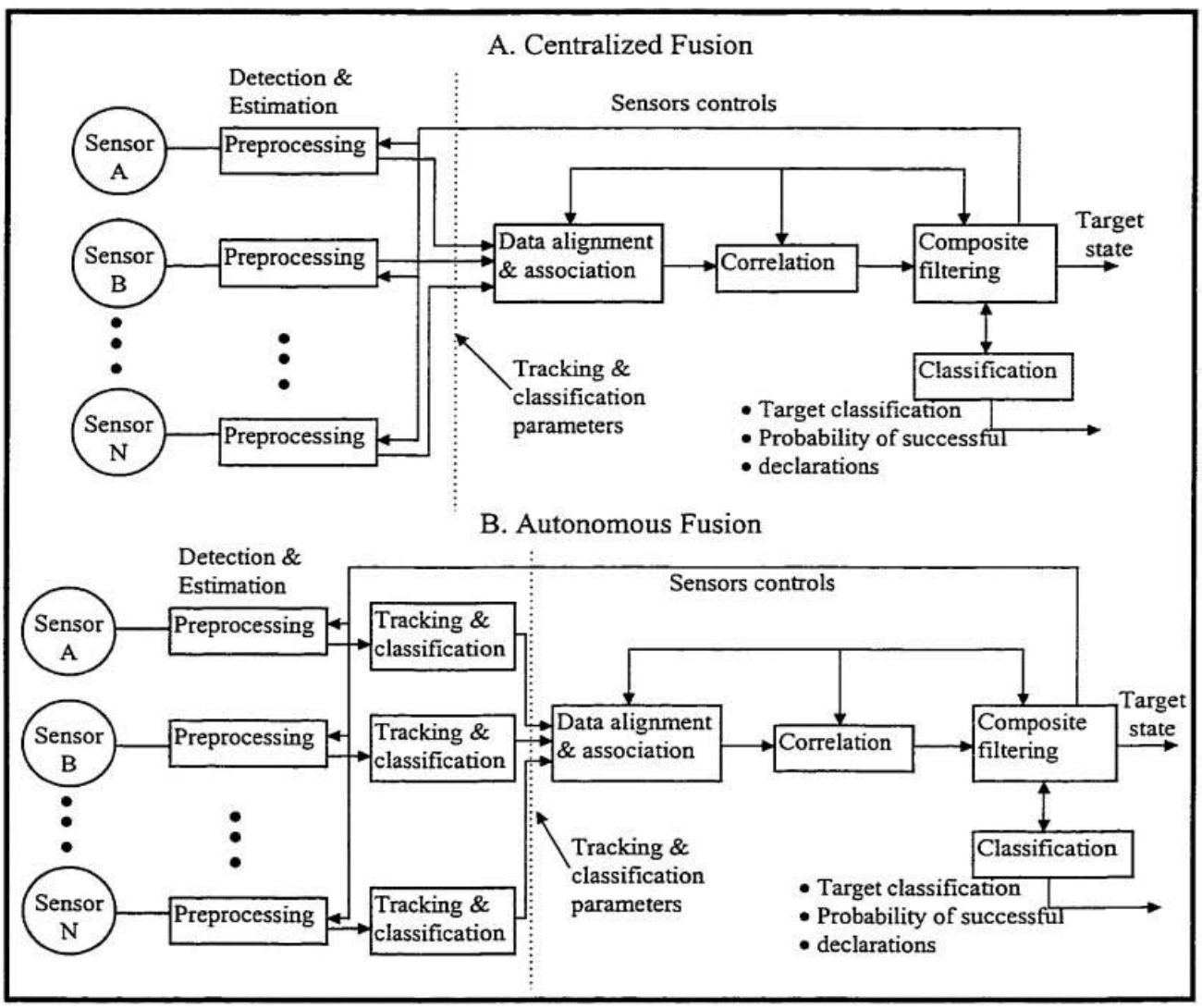


Figure 1.15(a): The architecture of centralized and autonomous sensor fusion [2].

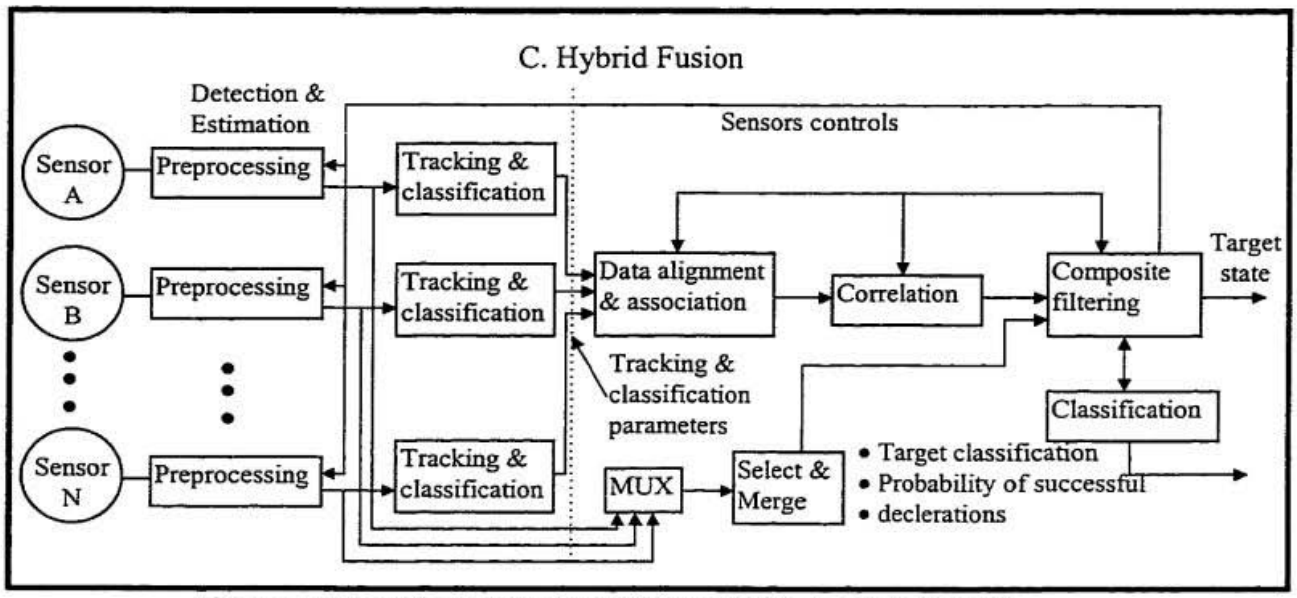


Figure 1.15(b): The architecture of the hybrid sensor fusion [2].

A network structure as shown in Fig.1.16 has been proposed as a framework for sensor management [16]. This framework is not supported with mathematical formalism.

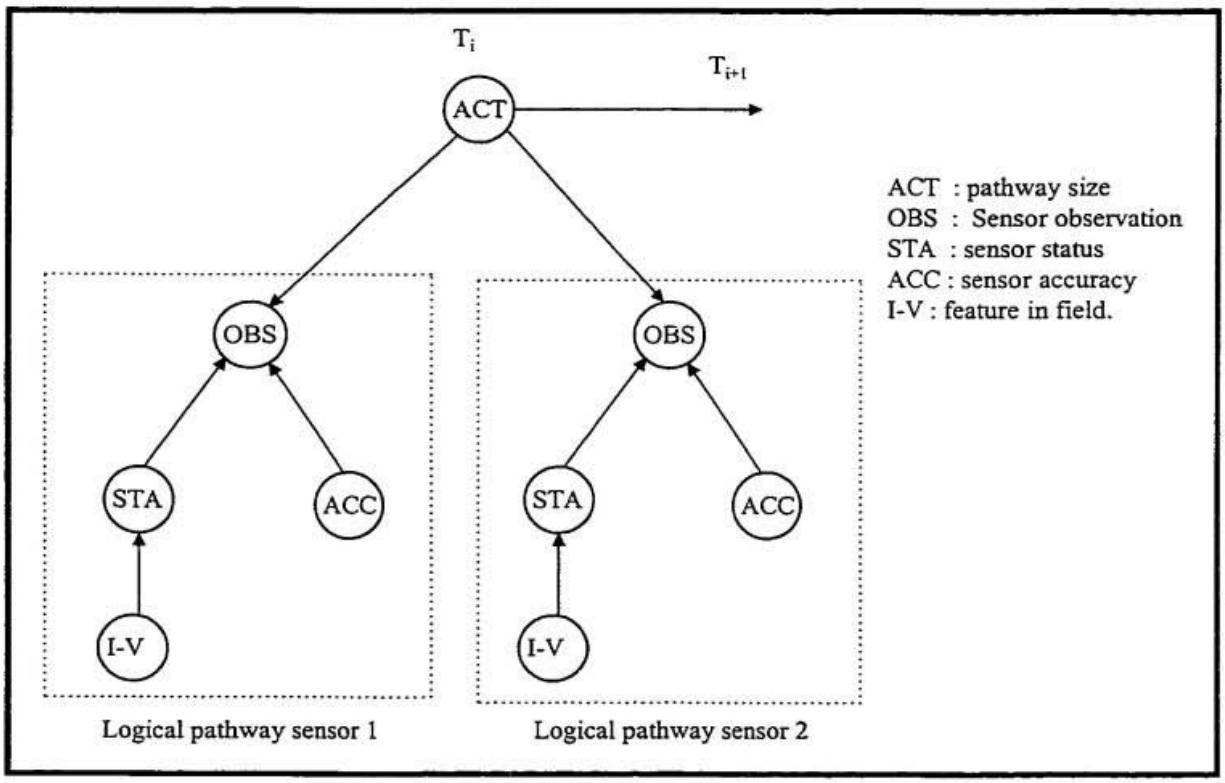


Figure 1.16: A network structure based sensor fusion framework [16].

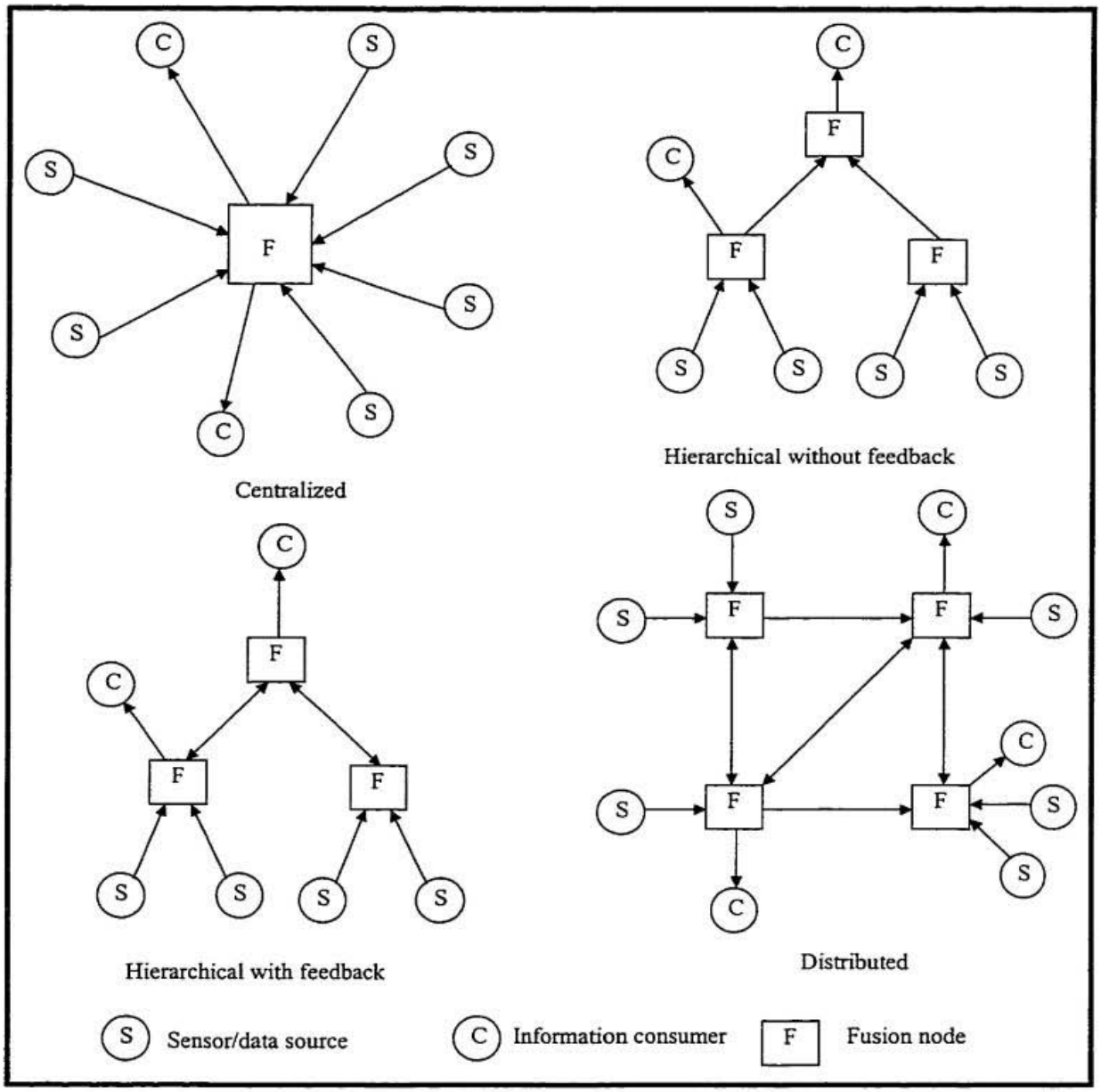


Figure 1.17: An information flow graph based sensor fusion architecture [17].

An architecture based on information graph for modeling the information flow in distributed fusion environment has been reported as shown in Fig.1.17 [17]. Despite the use of graph theory approach to represent different scenarios of sensor data integration, this architecture does not provide a mathematical formalism to define different modes of sensor data integration (e.g., independent, complementary, redundant).

An architecture for self-improving multisensory fusion system has been reported [5] as shown in Fig. 1.18. This is an adhoc graphical representation of a concept.

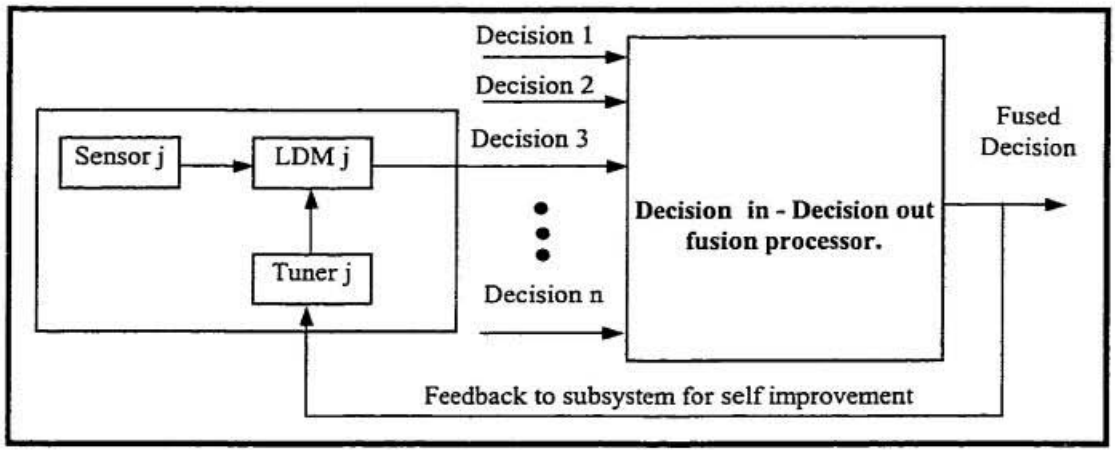


Figure 1.18: A self-improving multisensory fusion system architecture [5].

A sensor-level tracking approach with directed lines to indicate the flow of information has been used to combine data from multiple sensors for surveillance and tracking problems that arise in aerospace and defense as shown in Fig. 1.19 [18]. This sensor data combination lacks in mathematical formalism to model the sensor data integration scheme. It has been reported that the design and implementation of automated systems requiring fusion of data from multiple sensors are not well understood [18]. A general pattern of multisensor data integration using directed lines without the support of mathematical tools to simulate the system performance has been reported [19]. This general architecture of sensor fusion in intelligent systems is shown in Fig. 1.20.

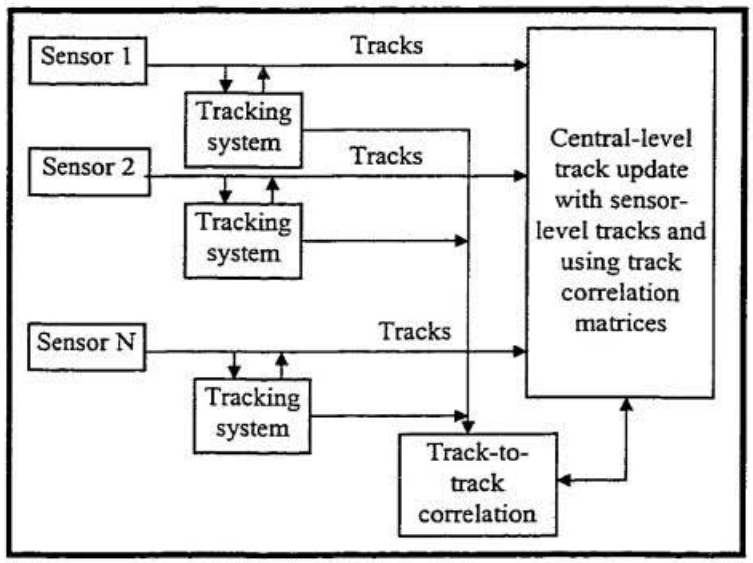


Figure 1.19: Sensor-level tracking approach to combine sensor data [18].

The uses of ad hoc approaches to integrate data from multiple sensors have been reported [20]-[27]. These approaches are mainly based on informal drawings and textual descriptions without the support of mathematical formalisms to model the data integration scheme. Despite the increasing dependence of our society on multi-sensori sensing system based intelligent systems, it has been mentioned that the development of sensing systems comprising different types of multiple sensors is still more of an art than a science [28].

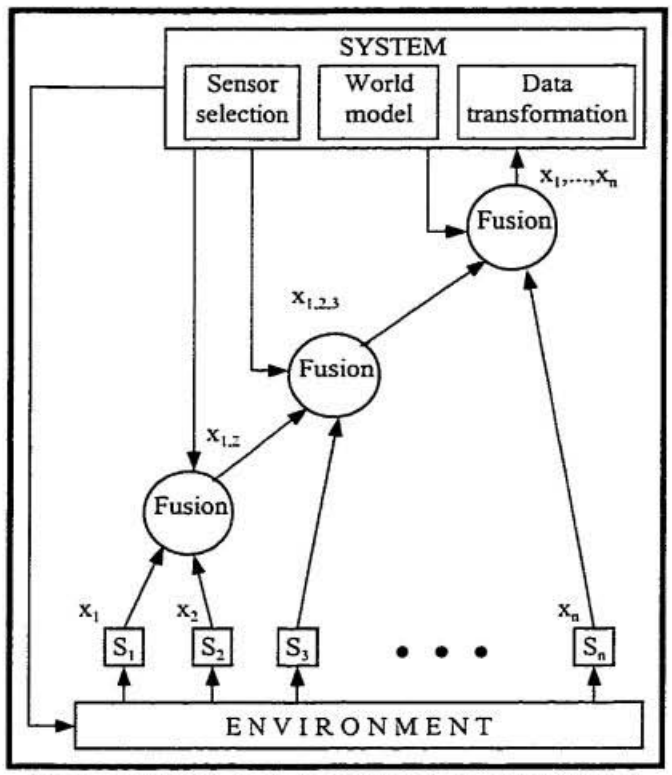


Figure 1.20: A generic pattern of multi-sensori integration and fusion system [19].

A mathematical framework called geometric feature relation graph (GFRG) has been proposed to integrate features sensed by different sensors [29]. This GFRG graph deals with algorithmic aspects of sensor fusion. The computational framework proposed in [30] deals with the formal descriptions of static nature of spatially distributed sensor networks. The schemes reported in [29] and [30] have deficiency in the use of system approach.

From the literature it appears that present state-of-the-art for engineering a sensor fusion system is deficient in using mathematical formalism. The architectures reported in the public domain as reviewed here are simple graphical representations of concepts for the integration of multi-sensori data. None of these architectures can accommodate different modes of sensor data integration: competitive, complementary, independent, and temporal in the same SFS. These architectures are not supported with mathematical formalism so that the different attributes of the system can be verified through simulation (e.g., logical and temporal correctness). These reported works do not provide the

framework for seamless implementation of these high level architectures. Although safety and mission critical applications have a high demand for multi-sensori systems, none of these architectures include fault-tolerance. The reliability profiles of different levels of fusion (e.g., data, features, decision) have not been addressed by any of these architectures. Therefore, based on the state-of-the-art review, there is a need to develop a comprehensive framework to address these issues for the development of highly reliable sensor fusion system.

Knowledge relating to this problem is available in the public domain and may be applied in searching for a solution to this problem. However, this knowledge is not directly usable as solution. As a matter of fact, some of the unique aspects of this problem have not been addressed by those developments. Some existing knowledge has helped to comprehend this problem, while some has been used as a tool to formulate the solution. The relationship of existing knowledge to the comprehension and formation of the solution proposed in this thesis is outlined in the following sub-sections. It is important to note that the objective of this literature review is not to give an exhaustive account of developments of related fields, but rather to describe how those developments can be used as aids to address this problem.

1.3.1 Sensor Fusion Sub-System and Petri nets

An SFS can be abstracted as a finite state machine. The discrete events, both periodic and aperiodic (stochastic), cause the SFS to change its state. As SFS is a real-time system, the state changes must satisfy stringent timing constraints. That is, one must guarantee that required computations be completed before specified deadline [31]. The dynamic interactions of those events go beyond the intuitive capability of the developer. It is required to equip the developer with a mathematical tool that is simple, intuitive and quantitative, so that the dynamic nature of the system can be readily represented graphically and through mathematical analysis. The developer will be able to perform a check of the properties related to the behavior of the SFS (e.g., precedence relations

amongst events, freedom from deadlock, repetitive activities, time required to serve a particular event). The simulation-based model validation should produce only a limited set of states of the modeled SFS, and thus should show the presence of errors in the model.

Petri nets are being developed in a search for natural, simple, and quantitative methods for modeling the behavior of the DES [1], [32]. Petri nets are system engineering tools. The analysis of the modeled system using Petri nets reveals important information about the structure and dynamic behavior of the modeled system [33]. In this modeling paradigm, the system is decomposed into interacting components (in an SFS, these are computing components). The modeled system changes its states through the generation of discrete events. These events are served through the interaction of corresponding computing components. This Petri net model of the system can be analyzed to check the precedence relations among events (for periodic events), deadlock, sequence of interaction of components to serve events, required time to serve an event. The randomness of aperiodic events can be addressed using stochastic Petri nets [34]. Petri nets have been used to design the simulator for flexible manufacturing systems (FMSs) and to verify the presence/absence of deadlock of the logic used to design the hardware/software for the controllers used in the FMSs [35]. The modeling of a real-time system specification to determine whether the specification is schedulable with respect to the imposed timing constraints has been performed with Petri net models [36]. The ability of Petri nets to evaluate the performance of real-time systems has been demonstrated in the literature [37]-[39]. From the study of the literature, it appears that Petri net can be a useful tool to engineer an SFS.

1.3.2 Discrete Event Requirements (DEVR)

The performance models evolve from descriptions of the system performance at the total system level to component properties in the late design cycles. The use of same mathematical formalism at different levels of system development including the

requirements phase has been outlined in [40]. This work has not proposed any quantitative method in this aspect. Time constraint discrete event formalism has been used to ensure end-to-end requirements of real-time systems [41]. This work has structured the system under development as a set of process components connected by asynchronous channels, in which the end points are the system's external inputs and outputs. Although this work includes a mathematical formalism, it lacks simplicity. A graphical tool with textual descriptions has been used for requirement specification for process control system [8]. Although this method has tried to represent the discrete event interaction of a process control system in graphical format, due to the lack of mathematical formalism this technique does not provide the means to analyze the model to verify its correctness and completeness. A process algebraic approach has been used to model the requirements of resource-bound and real-time systems [42]. Due to a complicated mathematical formalism and the lack of a graphical representation, this method may result in poor communication between the developer and the client.

It has been reported that it is a challenge to find suitable mathematical theories and notations that allow a designer to record the deep insight in the properties of the controlled physical process [10]. Ambiguous textual descriptions or mathematical notations that the clients find difficult to understand will impede the development process of SFS. The requirement model should be simple, natural and must have mathematical formalism for analysis. The requirements of the SFS can be abstracted as DES. Therefore, Petri nets based virtual machine modeling of the requirements of the SFS will be an effective solution to this problem. The developer will be able to analyze this model to ensure that the system requirement model satisfies required system functional goals and temporal constraints. Having a natural graphical property, this Petri net model will enable the requirement engineer to communicate with the customer easily. Moreover, this requirement model can be decomposed into subsequent phases of the development to model the interaction of discrete components. Therefore, use of the same mathematical model of the system at different levels of development will minimize the flow of errors from one level to the next.

1.3.3 Discrete Event Specifications (DEVS)

Recent developments in the paradigm of advanced robotics and intelligent automation has shown how systems may be advantageously represented as discrete event models by employing techniques based on the DEVS formalism [43]. This DEVS formalism is a means of formal representation of discrete event systems capable of mathematical manipulation, just as differential equations serve this role for continuous systems. The DEVS formalism is a set of models together with operators that combine models to form other models in ways in which real systems are connected [44]. The use of DEVS formalism to measure the performance of DES has been demonstrated [45].

There have been developments in the field of discrete event real-time system specifications using approaches other than DEVS formalism [42],[46],[47]. Usually these developments use either complex mathematical notations or textual descriptions. As a result, they are deficient either in simplicity of representation or analytical ability. Therefore, these developments will not be very useful for the specification of the SFS.

Petri nets being graphical and mathematical DEVS formalism provide a suitable environment for modeling and analysis of the specification of the SFS. Moreover, since Petri net formalism has been selected for modeling the requirements of the SFS, there will be a uniform transition of the development from requirements to specification phase. The importance of the use of the development process as the stepwise reduction of abstraction has been reported [40]. Petri net has been used for modeling the automation system in hierarchical and modular fashion [48],[49].

1.3.4 Reasoning About the DSPU Architecture (RDA)

The development effort in avionics has resulted in an integrated modeling approach in embedded computing system development [40]. The goal of this approach is to abstract the system under development at different levels of complexity. The top most level is the model of system requirements from user point of view. The subsequent levels steadily reduce the abstraction from conception (system requirements) to implementation (physical system). The DSPU architecture will implement the DEVS model of the system. Therefore, the reasoning about the architecture of the DSPU should be based on the DEVS model. The use of a system level approach has been recommended to address this problem [15]. This work [15] has reported that the lack of system level viewpoints and approach of developing embedded computing systems are the greatest cause of the problems experienced when computers are used to control complex processes.

According to the DEVS model, the DSPU is supposed to respond to both periodic and aperiodic events within specified time windows. Scheduling of real-time systems has addressed to be the problem of periodic events[50],[51]. There is a need to develop a stochastic process model to address the aperiodic events. Then this process model should be used to reason about the underlying architecture of the DSPU. The reasoning of the architecture should be based on the quantitative analysis of the DEVS model and the performance of the processing modules, so that the temporal specifications of the events can be satisfied. To facilitate the development process the Petri net can be used as a mathematical tool to model this reasoning process.

1.3.5 Fault-Tolerance of the SFS

To show high reliability, the acceptable probability of failure of the SFS is very small, typically in the range of 10^{-4} to 10^{-10}, depending on the consequences of the failure [52]. For a SFS to be adequately reliable for safety and mission critical operations, it must be capable of surviving a specified number of random component faults with a probability approaching unity. The use of component level redundancy has been suggested to achieve this objective [53]. Due to the stringent real-time requirements and costs, the redundancy management is an important issue to consider. It has been suggested that the overhead associated with managing redundancy must be quantified precisely so that certain guarantees about the real-time behavior of the system can be made [54]. The failure of components generates aperiodic events. Therefore, a stochastic Petri net model will help to address this problem [55].

The fault-tolerance techniques available in the public domain suspend the operation of the system during the fault clearance time. If this type of technique is used to address the fault tolerance of the SFS, data from the sensor during the fault clearance time will be lost [56]. Therefore, there is need to look at this problem to adopt fault-tolerance in the development of the SFS.

If a sensor fails or partially malfunctions and its effect is not considered in the fusion of sensor data, the dependability of the fused information will suffer. While one of the solutions may be to use redundant sensor system [57], such a sensor system cannot detect all types of faults (e.g., transient faults). Moreover, in the operation of unsupervised intelligent systems in mission critical operation (e.g., autonomous deployment of scientific experiment in space) redundant fault-free sensors may not be available to detect and replace a faulty sensor. In such an operational scenario, even a partially faulty sensor may be required to continue functioning. Therefore, there is a need to develop a scheme to address this problem. The solution should be simple and adaptable.

1.4 Approach of This Thesis Work

The SFS is an embedded real-time computing system. The importance of the use of system level approach to develop this type of system has been reported [15]. The tools supporting system level approach should be simple and mathematical. The simplicity will enable the developer to comprehend the system to avoid errors in different phases of development. The mathematical attributes of the tool will equip the developer to verify the conformity of the functions of the system under development with the customer's requirements.

The adopted approach should be a stepwise reduction of abstraction to simplify the complexity of development process [40]. This should produce a smoothly evolving set of designs at different levels of abstraction. The design approach should proceed from conception to implementation in a cyclic manner.

The simplicity and the mathematical nature of Petri nets have already been mentioned. Using the Petri net model it is possible to decompose the complexity of the system in hierarchical fashion. It has been explained that the Petri net formalism can be used in different phases of developments. Therefore, stepwise reduction of abstraction of development can be realized through this system modeling formalism.

The Petri net based DES modeling formalism has been adopted in this thesis to approach this problem. This approach decomposes the problem in a hierarchical fashion. In each cycle, the level of abstraction is reduced. In the early cycles, the abstract model may be a combination of hardware and software, but in the late cycles, they are very specific hardware and software design representations. In every phase, the developer will be able to verify the correctness of the functionality of the system under development.

1.5 Objective of this Thesis

The objective of this thesis is to synthesize engineering knowledge to develop a highly reliable SFS using a system engineering approach. Although the main focus of this thesis is to address the system aspects of the development process of the SFS, due to insufficient public domain solutions, this thesis also includes the unique algorithmic aspects pertinent to the development of highly reliable SFS for mission and safety critical applications (e.g., space, medicine). The long-term objective of this thesis is to develop a software tool to automate the development process of reliable sensor fusion systems. The specific short-term objectives are summarized in the following points:

1. To develop formalism for the DEVR model of the SFS with the provision of different modes of sensor integration in the same system: competitive, complementary, independent, and temporal. This intuitive and quantitative model of the requirements will be a virtual machine to satisfy the customer needs. This model will be the reference of communication between the customer and the development engineer. The intuitiveness of this model will help the clients to understand whether the system under development satisfies their requirements. The quantitative aspect will enable the developer to analyze different attributes of the system (e.g., modes of sensor data integration, logical and temporal correctness).

2. To develop a formalism for the DEVS model of the SFS. This model will enable the developer to define the dynamic interaction among the computing components to serve an event. The event level specification will be decomposed into the computing component level. The allocation of time for the computing components to serve a set of periodic and aperiodic events will be optimized through this model. This model will help the developer enhance the logical and temporal correctness of the execution of computing components to serve the events.

3. To develop reasoning taxonomy of the DSPU. This taxonomy will establish a link between the DEVS model of the system and the underlying computing system to implement this model. The quantitative aspect of this reasoning process will enable the developer to design the architecture of the DSPU to satisfy the DEVS model.

4. To develop a framework to measure the reliability that data will be provided to each stage of sensor fusion from the supporting sensors. It is also an objective to develop a predictable redundancy management scheme. This scheme will enable the developer to consider the effect of redundancy management overhead on the system performance. This quantitative information will be considered in the DEVS model of the system, so that the architecture of DSPU will keep enough room (determined from redundancy management overhead) to cope with the aperiodic events while maintaining logical and temporal correctness of DEVR model.

5. To develop a simple and adaptable sensor fault detection scheme, so that sensor fault-tolerance can be implemented using estimation based fault-detection approach. Due to the potential of transient faults to corrupt sensor data in safety and mission critical operations (e.g. medicine and space), one of the objectives of this work is to detect and locate transients in sensor data stream.

6. To develop methodology to minimize the effect of lost sensor data during fault clearance time. This work will address the problem of data recovery during the fault clearance time. Appropriate algorithms will be developed to recover this data. This data recovery scheme will also extend the data acquisition time. An estimation of this extension will be provided which should be considered in developing the DEVS model of the system.

1.6 Overview of this Thesis

The synthesization process of engineering knowledge to satisfy the objective of this thesis has tailored the relevant developments in the related fields to fit in the solution domain. This development can be broadly classified as the design automation for highly reliable sensor fusion systems. The theoretical developments of this work have been reported in main body of this thesis. The verifications of these theoretical developments through simulations and experimentations have been summarized in the Appendix.

The development of the formalism to capture the customer's requirement in terms of a discrete event dynamic virtual machine known as DEVR model is reported in **Chapter 2**. This proposed modeling technique has the ability to model different modes of sensor data integration: competitive, complementary, independent, and temporal. The graphical representation and quantitative analysis of this model is also shown. The quantitative analysis justifies the logical and temporal correctness. The realization of the task directed sensing and abstraction of RS as a sensor in the DEVR model to provide bi-directional communication are depicted in this chapter. The quantitative framework to measure the utilization factor of the underlying computing system is developed to realize a cost-effective SFS. It is also shown that this model can be analyzed to check the presence of deadlock, reachability, and repetitiveness of the operation of SFS.

Chapter 3 reports the development of the formalism of the DEVS model of the system. The dynamic interaction among the computing components to realize the DEVR model of the system under development is shown in the DEVS model. The optimization model for the allocation of time to computing components considering both temporal and computational constraints is reported in this chapter. The sensitivity analysis of component execution time is reported here to detect temporally critical computing components. This detection will help further development and/or special implementation (e.g., implementation in hardware) of these critical components.

The reasoning taxonomy of the DSPU is developed in **Chapter 4**. The relationship among service time of an event, waiting time in the queue, and the execution time and parallel nature of the corresponding traces is used as the basis of reasoning for the architecture of the DSPU. The reasons of randomness of execution times of the computing components on modern processors are reported here. Necessary guidelines are also provided to avoid this randomness of execution time to realize reliable implementation of DEVS model on the underlying embedded computing system.

The potential to enhance the reliability of the sensing system using redundant sensors is evaluated in **Chapter 5**. A novel technique to measure the probabilities of failures of different levels of sensor fusion (e.g., data, features, and decision) due to the failure of the supporting sensors is proposed in this chapter.

The development of the estimation technique using local statistics to detect and locate sensor's faults (specifically transient faults) is reported in **Chapter 6**. The profiles of local statistics of four test signals at transient faults are also evaluated. The dependence of local statistics based approach on the location of transient, window size, the location of window relative to the starting of transient and the frequency of the transient to detect and locate transient fault are evaluated through simulation.

Chapter 7 reports the development of the restoration scheme of lost sensor data during the fault clearance period. The reported scheme minimizes the loss of these real-time sensor data during fault-clearance period. This scheme is based on the restoration of data through parallel sensing. The restoration processes for both dual and triple modular redundancy schemes are explained. The effects of both hardware and software implementation of voting logic on the performance of the system and the quality of restoration are shown, and it is shown that this scheme is capable of recovering almost every datum lost during fault-clearance.

Chapter 8 summarizes the contributions and provides recommendations for future work.

1.7 The Novelties of this Thesis

This thesis presents a novel, unified framework for the development of reliable sensor fusion system, which will help produce a set of designs reducing the system abstraction from conception to implementation. The proposed framework allows the developer to avoid faults in both the development and operation phases of SFS's life cycle. The following points summarizes the novel aspects of the work proposed in this thesis:

1. The proposed DEVR model of the SFS provides a novel framework for modeling the system requirement as a virtual machine covering different modes of sensor data integration. The quantitative attributes of this framework enable the developer to analyze different aspects of the system under development at different phases of the development process to ensure that the system satisfies its intended purpose.

2. The proposed DEVS model of the SFS allows the developer to decompose the DEVR model in a hierarchical fashion to the computing component level. This unified approach helps ensure that the dynamic interactions among the computing components satisfy the logical and temporal correctness of the DEVR model.

3. The proposed derivation of the architecture of the underlying computing system from the DEVS model ensures that temporal correctness of DEVR model is provided in the operations phase of SFS. The identifications of sources of randomness of execution times to run computing components on modern processors and the proposed solutions to avoid them allows the developer to implement an SFS ensuring temporal correctness in operation phase.

4. The measure of reliability that data are provided at different levels of sensor fusion by the supporting sensors helps the designer to measure system performance at different levels of sensor integration. The identification of computing overheads to incorporate

hardware redundancy and proposed different techniques to cope with this overhead helps to develop predictable redundancy management schemes.

5. The proposed local statistics-based approach is a simple and adaptive sensor fault detection technique. This technique has the ability not only to detect a transient fault at different conditions, but also to locate the fault.

6. This work proposes different schemes to restore time-critical sensor data lost during fault-clearance interval. This contribution helps develop fault-tolerant sensor fusion system for safety critical operations where the system performance degrades considerably due to loss of data.

To the best of the author's knowledge there is no report of such developments in the public domain. It appears that these novelties have sufficient potential to enhance the state-of-the-art of engineering methodology for developing reliable sensor fusion system for safety and mission critical applications.

Chapter 2

The Discrete Event Requirements Model of the Sensor Fusion System

2.1 Introduction

From system point of view, the basic unit of the user's need can be defined as the requirement of completion of a particular sensing task within a specific period of time. A block diagram representation of the requirement for the measurment of the standard deviation of N samples of a vibration sensor data is shown in Fig. 2.1. This can be clarified as follows: if the inputs are ready, the enabled task should be executed within the specified time. The enabling of a task can be considered as an extra input in addition to the necessary data. A Petri net formalism for modeling this requirement is shown in Fig. 2.2. The availability of the inputs is the generation of an event and the event is served by the execution of the corresponding process P_r within time Te. The execution time includes both the processing of the inputs and the transmission of outputs to the corresponding buffers. Therefore, the basic unit of the user's requirements can be modeled as the service of an event if certain conditions are met. The outputs of an event can be used as the partial fulfillment of the generation of another event. The service of this event within the

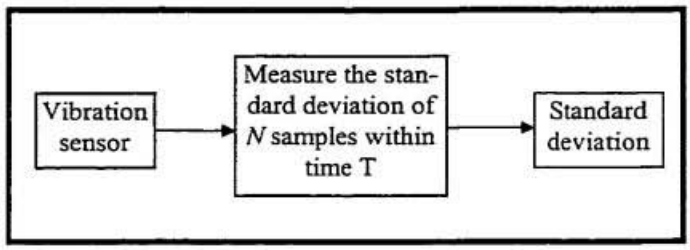


Figure 2.1: Block diagram of completion of a task.

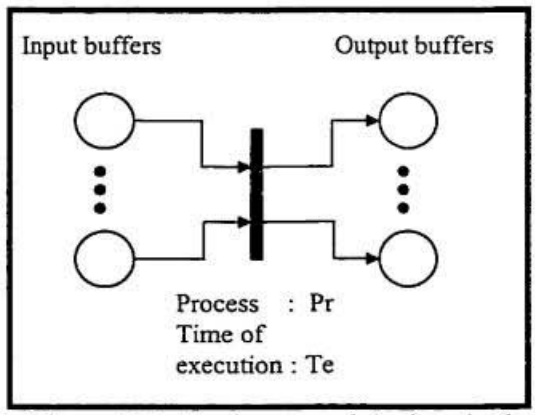


Figure 2.2: Petri net model of typical user requirements.

specified time window is also part of the user's requirements. Therefore, the user's sensing requirement can be abstracted as a discrete event dynamic system (DES). The discrete event requirements (DEVR) model of the SFS transforms the user's need into a finite state virtual machine. This virtual machine represents the dynamic interaction of the discrete events to satisfy the user's need. The user's requirements can be broadly classified as:

- Periodic requirements
- Aperiodic requirements

The growth in the complexity of the SFS creates numerous problems for the engineers. In the requirement analysis stage, one is required to deal with the increased capabilities of these systems due to the unique combination of hardware and software, which operate under stringent timing constraints. It is well known that the flaws in understanding requirements can substantially contribute to the reliability, time and cost [8]. The lack of a formalism contributes to the improper understanding by the developer of the user's requirements of these systems [8]. Statistics of non-clerical requirement error are shown in Fig. 2.3. Moreover, requirement management is an important issue. Therefore, the analysis of the DEVR model should address to answer the following questions to minimize these errors:

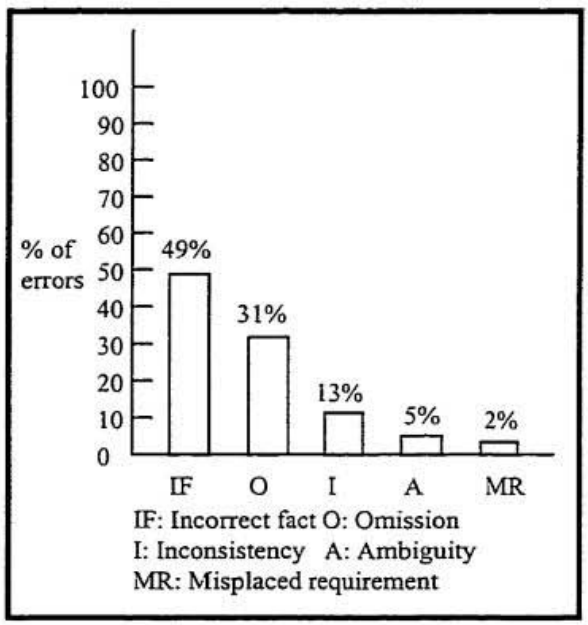


Figure 2.3: Types of non-clerical requirement error [©1981 IEEE Computer Society Press].

1. Logical correctness
2. Temporal correctness
3. Reachability

4. Presence of deadlock
5. Repetitiveness
6. Sensitiveness

2.1.1 Petri Net Model of Different Modes of Sensor Data Integration

The objective of sensor fusion is to integrate information from multiple sensors in order to make a more accurate or complete inference about a physical event, activity, or situation than could be achieved with a single sensor [2],[3]. In this fusion process, sensors can provide temporally related competitive, complementary, and independent information [6]. The Petri net models of different modes of sensor data integration are represented in the following subsections.

2.1.1.1 Competitive or Redundant Sensor Integration

Competitive integration requires replicated sensor readings, which ideally are identical, but in reality may be noisy and one or more sensors may fail partially or completely. It is the objective to ignore the erroneous information from faulty sensors in the integration process. A Petri net model of this integration process is shown in Fig.2.4. In this process model, data in the form of *n* tokens come from *n* competitive sensor processes to the place Pc. It has been assumed that tokens will be sent to the place only if the corresponding sensor functions properly. Availability of only one token in the place Pc is required to drive the following process Pr. Therefore, this sensor suite will serve the purpose as long as one sensor is functioning properly.

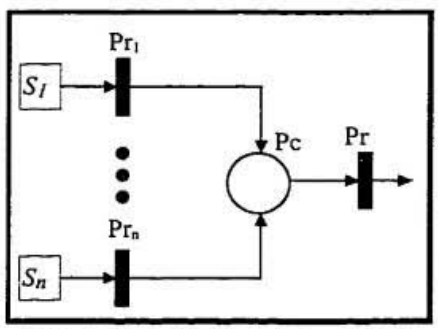


Figure 2.4: Petri net model of competitive sensor integration.

2.1.1.2 Complementary Sensor Integration

Partial and overlapping information from more than one sensor is integrated into more complete information. The synergistic use of overlapping and complementary sensor data provides information that is not available from individual sensors. The integration of information from *n* complementary sensors in the Petri net model is shown in Fig.2.5. Here, the process Pr integrates information derived from *n* sensors. Every sensor specific process sends only one token in the place Pc and *n* tokens are required in the place p to drive the integration process Pr. In this operational scenario, the integration process runs if every sensor functions properly.

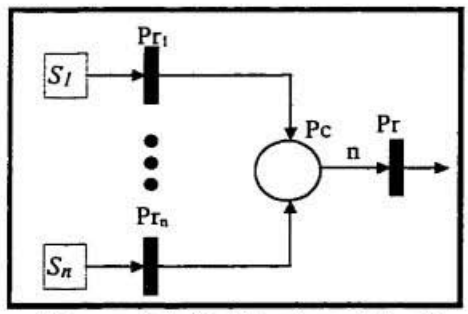


Figure 2.5: Petri net model of complementary sensor integration.

2.1.1.3 Independent Sensor Integration

Independent data from *n* sensors are integrated to develop wider world model of the operating environment. The Petri net model of such an integration scheme is shown in Fig. 2.6. Failure of one or more sensors will result in an incomplete model of the environment. This modeling technique can also be used to model the validation of diverse sensor data to provide robustness in the system at a fusion level (e.g., the use of laser ranging sensor to validate stereo image data).

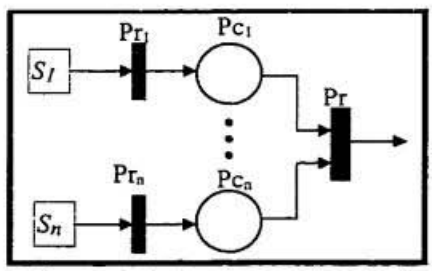


Figure 2.6: Petri net model of independent sensor integration.

2.1.1.4 Temporal Integration

In temporal integration, data sampled from different sensors at particular time intervals are processed to sense the environment in a particular sequence so that temporal

relationship among the sensed signals can be exploited. Using the frequency and phase concept data can be acquired from different sensors in a particular sequence to achieve this objective. The proposed mathematical formalism to model temporal integration is explained as periodic requirements in the following section.

2.2 Periodic Requirements

In SFS, data should be acquired from sensors at regular intervals of time. These events are generated using a clock signal as shown in Fig. 2.7. The system consists of a set of sensors, S=$\{S_i$: $i \in I$, *I is the size of the set*$\}$. Ts_i is the period of individual sensor. Min(ΔTs_i) is the allowable service time for the periodic events of the *ith* sensor. This is the time period between the generation of the event for the *ith* sensor and the generation of the temporally closest event for another sensor. The value of min(ΔTs_i) is dynamic as shown in Fig. 2.8. In the worse case, this value may be reduced to zero resulting in overlapping of sensing periods of multiple sensors. If this problem is not addressed in the development of the SFS, the single processor based SFS may not be able to show expected performance. Moreover, due to the dynamic nature of the problem, the performance will be unpredictable. As a result, the reliability of the system will suffer. One of the novelties of this work is to address this problem.

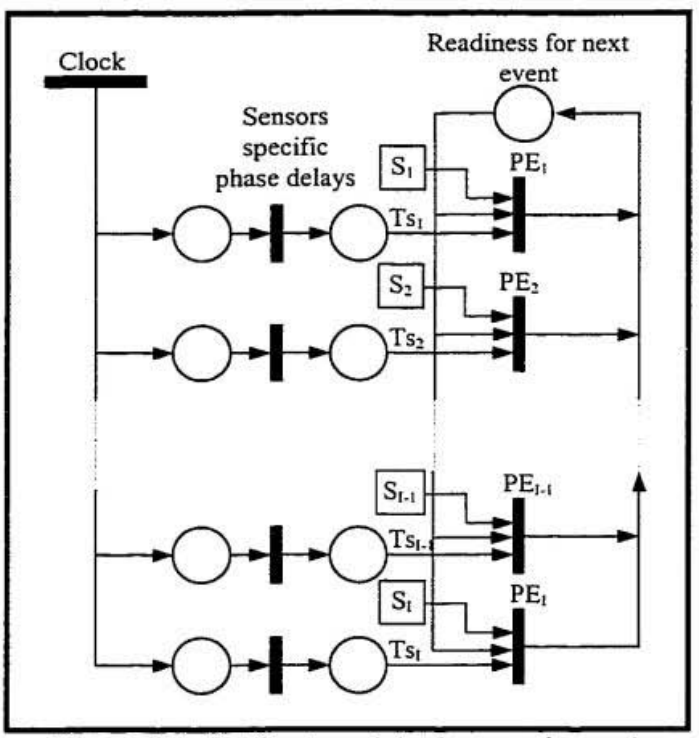


Figure 2.7: Petri net model of generation and service of periodic events for data acquisition.

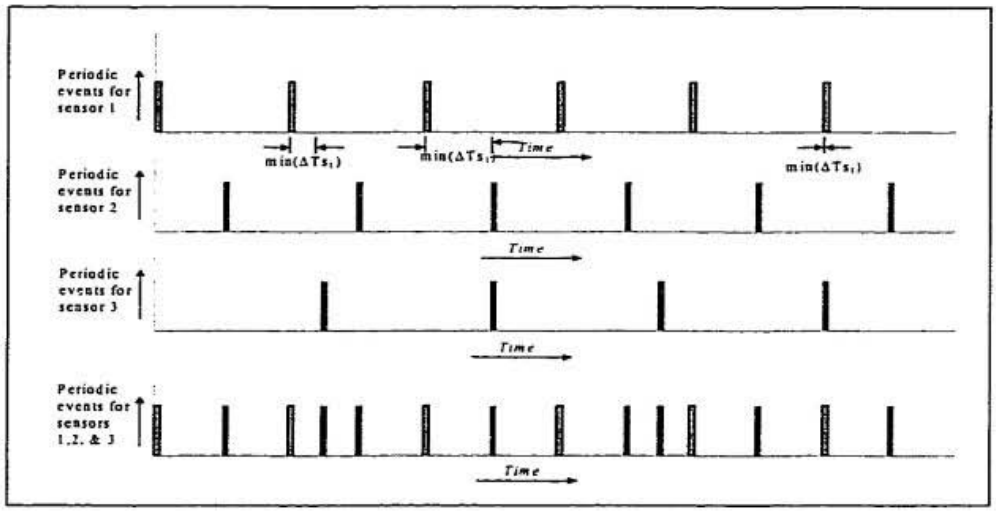


Figure 2.8: Timing diagram of the generation of periodic events for three example sensors.

To overcome the overlapping of sensing periods of multiple sensors, a scheme should be developed so that required service time can be guaranteed. A simple solution may be to reset the system, when the value of ΔTs_i goes lower than the acceptable limit. The time to reach this point can be called the system period, SP. There is a cost to reset the system in terms of quality of performance. In the worse case, the system may not function at all if the value of SP is smaller than the threshold limit. Therefore, it is necessary to devise an algorithm to maximize SP while guaranteeing the critical value of $\min(\Delta Ts_i)$. The problem can be defined by the following set of equations. Here, all the numbers are positive integer.

$$N_i = \Phi_i + \sum_{j=1}^{n} Ts_i \tag{2.1}$$

$$(N_i - N_j) \geq \min(\Delta Ts_i) \text{ for all } i \neq j \text{ and } N_j \geq N_i \tag{2.2}$$

$$\text{Maximum value of } N_i,\ N_{\max} = \int(\Phi_i, Ts_i) \tag{2.3}$$

$$\Phi_i^{\,l} \leq \Phi_i \leq \Phi_i^{\,u} \tag{2.4}$$

$$Ts^{i}{}_i \leq Ts_i \leq Ts^{u}{}_i \tag{2.5}$$

Now, the job is to select optimum values for Ts_i and Φ_i to maximize the value of N_i (i.e., the system period).

Here, N_i : the instances of generation of periodic events corresponding to the *ith* sensor.

Φ_i: the phase of the *ith* sensor.

Φ_i^l : the lower limit of the value of the phase of the *ith* sensor.

Φ_i^u : the upper limit of the value of the phase of the *ith* sensor.

T_i^l : the lower limit of the value of the period of the *ith* sensor.

T_i^u : the upper limit of the value of the period of the *ith* sensor.

The analysis of the computational complexity:

Let the difference between upper and lower bound of Φ_i be D_i and that of Ts_i be C_i. Therefore, the size of the solution set for I number of sensors is

$$D_1 \times C_1 \times D_2 \times C_2 \times .... \times D_I \times C_I \tag{2.6}$$

So, the sequential search of this solution set for the maximum value of N_i even for a few sensors may be unrealistic. Two examples shown in Table 2.1 and Table 2.2 will help to understand the structural nature of this problem to reduce the solution size.

Table 2.1: The sensing sequence of an example sensor system

Sensors	Period	Phase	Time of generation of periodic events
1	5	0	0, 5, 10, 15, 20, 25
2	10	1	1, 11, 21
3	15	2	2, 17
4	20	3	3, 23

Table 2.2: The sensing sequence of an example sensor system

Sensors	Period	Phase	Time of generation of periodic events																												
1	3	0	0			3			6			9			12			15			18			21			24			27	
2	8	1		1								9								17							25				
3	10	2			2										12										22						
4	12	3				3												15												27	

In Table 2.1, the events do not collide. Here, the periods are integral multiple of 5 (i.e., grain size of the periods is fixed). The phases of the periods increase at constant step. From a knowledge of number theory [58] it appears that under this circumstance the events will never collide if the phase is smaller than half of the period grain size.

The analysis of the data of Table 2.2 reveals that due to the absence of the integral multiplicity of the periods, it is virtually impossible to guarantee that the events will not collide. Therefore, the solution of avoidance of collision of events can be simplified by imposing the following constraints to select periods and phases for the sensors:

The periods and phases should be integral multiple of grain sizes *Gp* and *gp* respectively and *Gp* is the integral multiple of *gp*. The grain sizes *Gp* and *gp* are the smallest period and phase respectively. No two sensors will have the same phase, but they may have the same period. The phase of any sensor should satisfy the following condition

$$\Phi_i < Gp \quad \text{for all } i \tag{2.7}$$

For more simplification of this problem, it can be assumed that the $\min(\Delta Ts_i)$ for each sensor is the same and maintains the following relation

$$min(\Delta Ts_j) \geq max\{\, min(\Delta Ts_j) : i \in I\} \tag{2.8}$$

Then satisfaction of the following condition will guarantee that $\min(\Delta Ts_i)$ will be maintained for each sensor of the set.

$$gp \geq \min(\Delta Ts_i) \tag{2.9}$$

The imposed constraints will limit the size of the sensor set. The maximum number of sensors is equal to *Gp/gp*. The maximum value of *Gp* is limited by the variability and minimum values of periods of sensors and the minimum value of *gp* is limited by the required maximum service time of any event. In the worse case, if the event service time is greater than *Gp*, this scheme will not accommodate more than one sensor. These constraints simplify the solution significantly. From the literature it appears that in real life applications, the number of sensors assigned to a single SFS is on order of 10 and the service time is much smaller than *Gp* [18-30]. For larger number of sensors, multiple SFSs may be used to cope with this problem.

This proposed technique has been used successfully to detect overlapping sensing periods in the design of an example sensor fusion system in section B.6. The modification of the values of phases and periods of the sensors using this technique have been used to avoid this overlapping.

This new development has improved the state-of-the-art of the engineering of multiple sensors based system. This critical design decision is based on sound quantitative reasoning.

2.2.1 Task Directed Sensing

A recent study has revealed that the reasoning sub-system only needs that information which is directly in support of the current tasks that the intelligent system is trying to accomplish [59]. The concept of 'task directed sensing' or active sensing is a response to this operational requirements [59]. The task directed sensing can be incorporated in the discrete event requirement model by abstracting a physical sensor as a set of virtual sensors as shown below:

S_i={ S_{ij}: $j \in V_i$, V_i is the size of the set of the virtual sensors for the *ith* sensor}.

These virtual sensors have different phases and periods as per the requirements of variable sensing from the *ith* sensor. In the synchronization planning phase of the periodic requirements as explained in the previous sections, these virtual sensors will be treated as independent physical sensors, one of which will be in operation at one time.

2.2.2 Communication with the Reasoning Sub-System

The closed-loop Petri net model of an intelligent system as shown in Chapter 1(pp.1, Fig.1.1) depicts that the communication between the sensor fusion sub-system (SFS) and the reasoning sub-system (RS) is bi-directional. The DEVR model so far developed only incorporates unidirectional communication, from the SFS to the RS. From the perspective of communication from the RS to the SFS, the RS can be abstracted as a sensor in the DEVR model of SFS. This incorporation of an extra sensor does not change the general DEVR model of the SFS other than the addition of an extra virtual sensor. The propagation time of feedback from the RS to SFS should be considered to specify the phase and frequency of this virtual sensor.

2.3 Aperiodic Requirements

In SFS, periodic events collect information from the environment reading sensor values. If the acquired information satisfies particular conditions, the corresponding processes should be executed to fuse these data within defined time windows. These conditional requirements are known as aperiodic requirements. The generation of this type of requirement is known as an aperiodic event. These aperiodic events are serviced by executing corresponding processes.

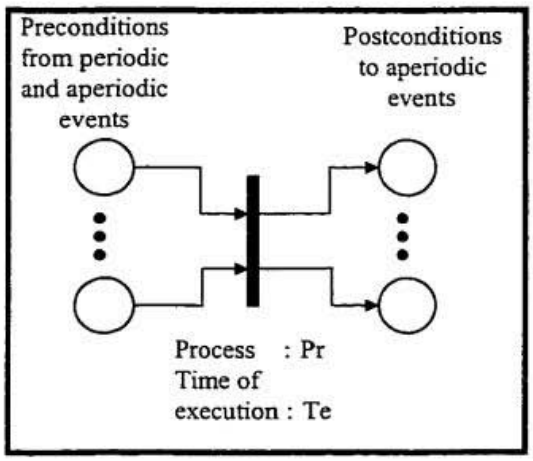


Figure 2.9: Petri net model of an aperiodic event.

A Petri net model of an aperiodic event is shown in Fig. 2.9. It should be noted that aperiodic events are dependent, while periodic events are independent. Conditions may flow to an aperiodic event from both periodic and aperiodic events. But an aperiodic event can generate conditions only for other aperiodic events.

An SFS consists of a set of aperiodic events, AE=$\{AE_i: i \in Z$, Z is the set of positive integers$\}$. The generation and service of these events are controlled by a set of preconditions, Prc=$\{Prc_i: i \in Z\}$ and a set of postconditions, Poc=$\{Poc_i: i \in Z\}$. If there are M preconditions and N postconditions, M-bit and N-bit binary numbers may be used to specify the corresponding preconditions and postconditions of an aperiodic event respectively. The value of a bit will indicate the presence or absence of the corresponding precondition or postcondition. Therefore, an aperiodic event can be defined by updating the basic definition of an event as shown,

$$AE=\{I, Prc, Te, Pr, Poc\} \tag{2.10}$$

Similarly, a member of the set of periodic events, PE=$\{PE_i: i \in Z\}$, can be defined as

$$PE=\{I, Te, Pr, Poc\} \tag{2.11}$$

Here, I is the identity number of the event; Te is the service time; Pr is the corresponding process.

2.4 Combination of Periodic and Aperiodic Requirements

The requirement of an SFS consists of dynamic interaction of a set of periodic and aperiodic events to derive a decision from sensor data. The periodic events are activated by temporal information. Therefore periodic events have no preconditions; but they have

postconditions and these are the subsets of preconditions of aperiodic events. The relationship between periodic and aperiodic events can be defined as

$$(PE_i.Poc \cap AE_j.Prc) \leq 1 \quad (2.12)$$

$$(PE_i.Poc \cap PE_j.Prc) = 0 \quad (2.13)$$

$$(AE_i.Poc \cap AE_j.Prc) \leq 1 \quad (2.14)$$

The life of a postcondition is Lpc_i. This finite life concept of a postcondition will enable the designer to incorporate the temporal value of information in the system model, which is an important factor to consider in modeling real-time systems.

2.5 Discrete Event Model of Requirements (DEVR)

The Petri net structure of the discrete event model of requirements, DEVR, is a four-tuple net, DEVR=(Pc, E, Prc, Poc). The tuple $Pc=\{Pc_1,Pc_2,Pc_3,....,Pc_m\}$ is a finite set of places to hold conditions, both preconditions and postconditions, $m \geq 1$. The number of the places should be at least one to hold the readiness state to respond to the next periodic event. $E=PE \cup AE$ is the set of all periodic and aperiodic events, $E \geq 1$. This means that the system has at least one sensor. The set of conditions and events are disjoint. The tuple *Prc* is the input function, which defines the required conditions for an event. *Poc* defines the conditions generated from the service of an event known as output functions. A place Pc_i holds the precondition of an event E_j if $Pc_i \in Prc(E_j)$; Pc_i holds the postcondition of an event Ej if $Pc_i \in Poc(E_j)$. Therefore, according to this model the most simple sensor fusion system may have only one sensor, a periodic event and no aperiodic event as shown in Fig. 2.10.

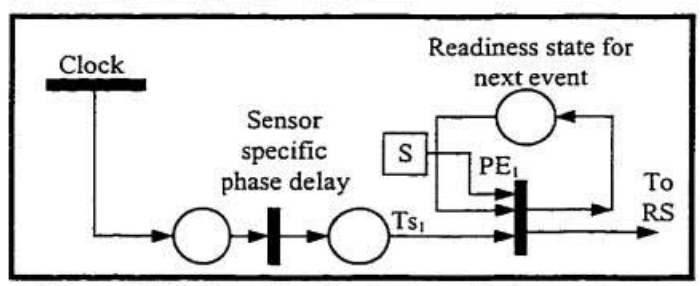


Figure 2.10: The simplest sensor fusion system.

This system only acquires data and after preliminary processing sends these data to the RS. The maximum number of sensors and aperiodic events are not limited by the model, that is rather defined by the user's requirements. The number of periodic events is equal to the number of sensors. In case of task directed sensing, each virtual sensor will have a periodic event. The DEVR model of a typical SFS is shown in Fig. 2.11. Here, all the aperiodic events are shown as a single event. The process corresponding to this event is responsible for the data fusion, feature fusion and decision fusion. After the formation of this DEVR model, the input and output functions of the modeled SFS can be studied to ensure that the system has been modeled according to this formalism. This study has

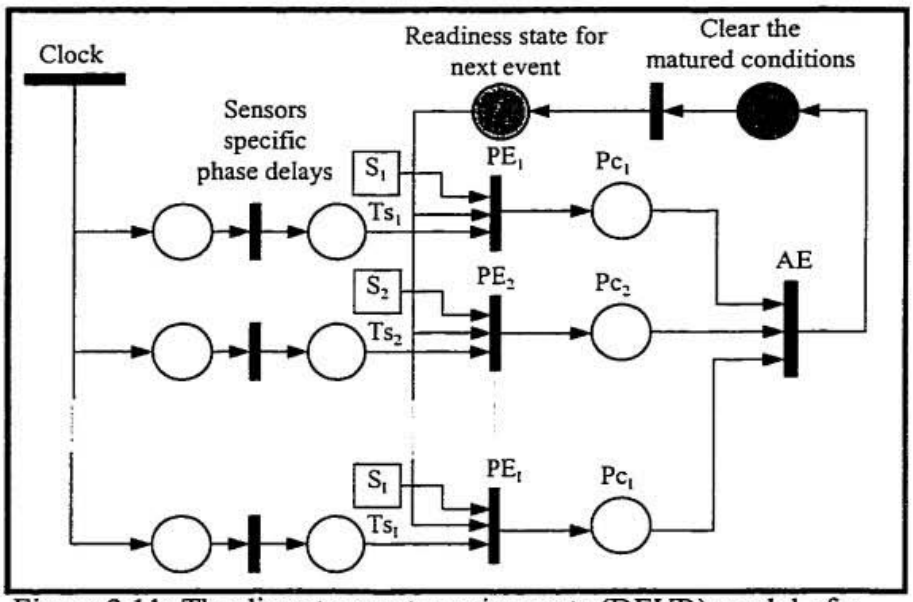


Figure 2.11: The discrete event requirements (DEVR) model of a typical sensor fusion system.

been performed on the model of an example sensor fusion system shown in section B.1. This characteristic of the proposed discrete event based modeling framework enables the designer to avoid partially faults at the very early stages of system development resulting in better reliability in system performance.

To make the concept comprehensible the following example can be considered. This sensing system consists of five sensors. The sensing job is done by the interaction of five periodic events and six aperiodic events. Seven places, $Pc_1,\ldots,Pc_7$, hold conditions

generated from periodic and aperiodic events. The post conditions generated by periodic events are shown by the bit pattern of Poc used in specification of periodic events. The bit patterns of Poc and Prc to specify aperiodic events define the required post conditions and generation of preconditions by the aperiodic events respectively. The DEVR model of this example sensor fusion problem is shown in Fig. 2.12.

Periodic events	**Aperiodic events**
87654321	87654321 87654321
$PE_1=\{\phi_1,Ts_1, Te_1, Poc_1=00000001\}$	$AE_1=\{1,Prc_1=00000011,Te_1,Pr_1,00100000\}$
$PE_2=\{\phi_2,Ts_2, Te_2, Poc_2=00000010\}$	$AE_2=\{1,Prc_2=00000010,Te_2,Pr_2,01000000\}$
$PE_3=\{\phi_3,Ts_3, Te_3, Poc_3=00000110\}$	$AE_3=\{1,Prc_3=00000100,Te_3,Pr_3,10000000\}$
$PE_4=\{\phi_4,Ts_4, Te_4, Poc_4=00001000\}$	$AE_4=\{1,Prc_4=00011000,Te_4,Pr_4,10000000\}$
$PE_5=\{\phi_5,Ts_5, Te_5, Poc_5=00010000\}$	$AE_5=\{1,Prc_5=00010000,Te_5,Pr_5,10000000\}$
	$AE_6=\{1,Prc_6=01100000,Te_6,Pr_6,10000000\}$

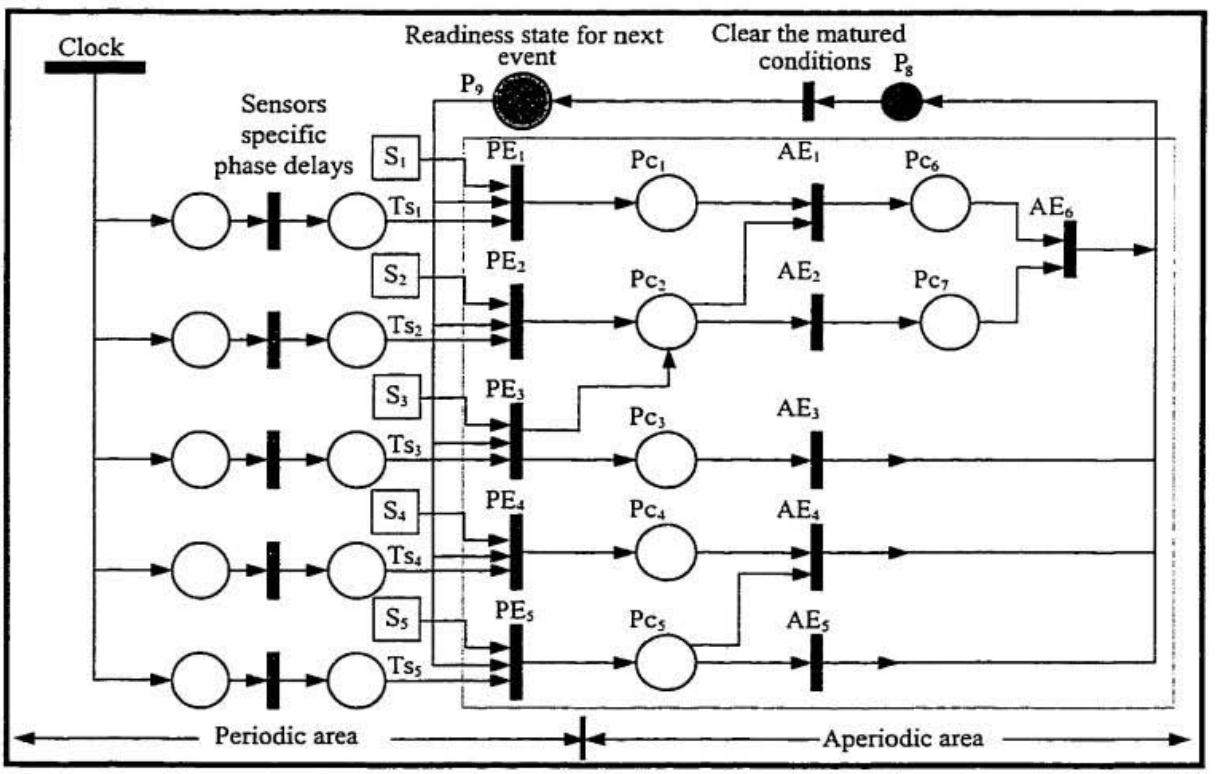


Figure 2.12: The discrete event requirement model of an example sensor fusion system.

2.5.1 The State Spaces of the DEVR Model

A marking μ is an assignment of conditions to the places of the model. Conditions are assigned to the places by the execution of the events and they reside in those places for their life times if they are not consumed by the events. At the end of the life-time they disappear from the model. The number and position of conditions in the model change during the execution of the system. The marking μ can be defined as an m-vector, $\mu=\{\mu_1, \mu_2, ..., \mu_m\}$, where μ_i corresponds to the number of corresponding conditions [1],[32].

The *state* of the modeled SFS is defined by its markings. The firing of an event represents a change in the state of the system by a change in the distribution of conditions in the system. The state space of the model with m conditions or places is the set of all markings, that is N^m, where N is the number of any conditions. The state of the system changes due to the firing of the events as well as due to the expiration of the life of the conditions.

During the life-time of the conditions, the change in state caused by firing an event is defined by a change function δ called the next-state function. When applied to a state μ and an event E_j this function yields the new state $\mu`$, $\delta(\mu, t_j) = \mu`$. The change of state due to the expiration of time of the conditions can be defined as $d(\mu, c_j) = \mu`$. c_j is the *jth* condition.

2.5.2 The Effect of Death of Conditions on the Performance of the Sensor Fusion System (SFS)

In the sensor fusion system, the conditions generated by both periodic and aperiodic events have a certain lifetime. If during this lifetime, the conditions are not consumed by

the aperiodic events, they will die. This provision is very useful to incorporate the temporal characteristics of conditions in the operation of the system, but this provision also creates complications in the system analysis. If it is assumed that no condition will die during the period $\min(\Delta Ts_i)$ for the *ith* event, the problem will be simpler. Under this operational scenario, the conditions can die during the time when the system is in readiness state. This will ensure that during $\min(\Delta Ts_i)$ the system will only change state due to the consumption and generation of conditions by the events. At the end of each periodic cycle the system will do housekeeping work to clear the conditions whose lives have expired. This assumption seems to be not unreasonable as $\min(\Delta Ts_i)$ is usually very small.

2.6 Analysis of Discrete Event Requirements Model

It has been shown that discrete event model captures the user's requirements as virtual machine. However, modeling by itself is of little use. It is necessary to analyze the modeled system. In order to verify the conformity of the modeled system performance with the user's requirements the independent (periodic) events are generated in the SFS at regular intervals of time. The phase and period information of the periodic events are shown by the following matrix

$$PET = \begin{bmatrix} \Phi_1 & Ts_1 \\ \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \\ \Phi_n & Ts_n \end{bmatrix} \tag{2.15}$$

The system starts at $t=0$. The flow chart of the algorithm for the generation of periodic events for $t < \Phi_i$ is shown in Fig. 2.13.

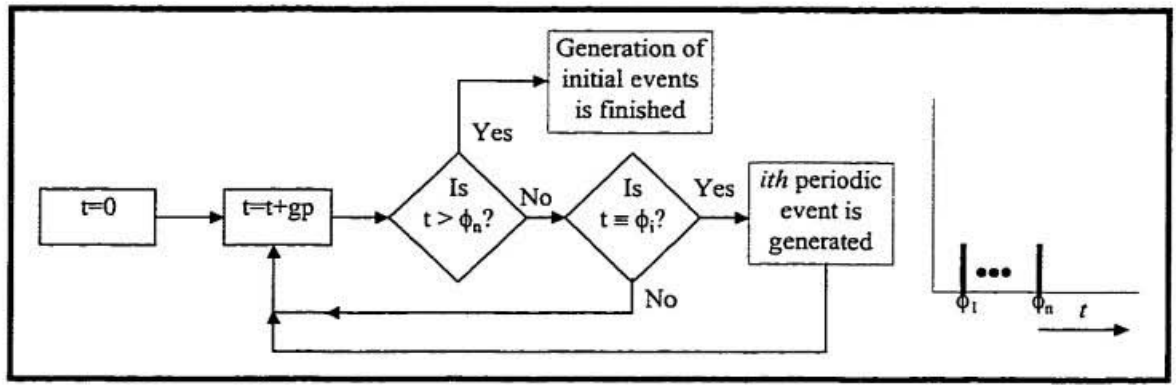


Figure 2.13: The flow chart of the algorithm for the generation of periodic events.

The generation of periodic events for $t > \Phi_i$ can be controlled by the following matrix relation

$$\begin{bmatrix} \Phi_1 \\ \cdot \\ \cdot \\ \cdot \\ \Phi_n \end{bmatrix} + \begin{bmatrix} Ts_1 & 0 & \cdot & \cdot & 0 \\ 0 & Ts_2 & \cdot & \cdot & 0 \\ 0 & \cdot & \cdot & \cdot & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & \cdot & \cdot & \cdot & Ts_n \end{bmatrix} \begin{bmatrix} Cs_1 \\ \cdot \\ \cdot \\ \cdot \\ Cs_n \end{bmatrix} = \begin{bmatrix} \Phi_1 + Ts_1 Cs_1 \\ \cdot \\ \cdot \\ \cdot \\ \Phi_n + Ts_n Cs_n \end{bmatrix} \tag{2.16}$$

If ($t=t+GP$) is equal to ($\Phi_i + Ts_i Cs_i$), *ith* periodic event is generated and Cs_i is incremented by one. Here, Cs_i is the number of cycles already generated corresponding to the *ith* sensor.

2.6.1 Logical Correctness

The firing sequence of the Petri net model can be used to check logical correctness. In fact, logical correctness can be verified by depicting the interaction between the events. The matrix representation of the postconditions and preconditions of the aperiodic events and postconditions of the periodic events make this task simpler.

Aperiodic

events	Preconditions	Postconditions
AE_1	$b_{11} \cdots b_{1m}$	$b_{11} \cdots b_{1m}$
$\vdots$	$\vdots$	$\vdots$
AE_n	$b_{n1} \cdots b_{nm}$	$b_{n1} \cdots b_{nm}$

Periodic

events	Postconditions
AE_1	$b_{11} \cdots b_{1m}$
$\vdots$	$\vdots$
AE_n	$b_{n1} \cdots b_{nm}$

The events responsible for the generation of an aperiodic event can be found by checking the value of the elements of the preconditions and postcondition matrices.

Theory:

For the *ith* aperiodic event, if the b_{ij} element of the precondition matrix is *1*, the events having the *jth* element of postconditions matrix of non-zero value are responsible for the generation of the *ith* event.

The numerical nature of the representation of system behavior makes software based automation feasible for this task. This automation has potential to make designer's job faster and more accurate.

This concept can be explained by analyzing the previous example problem.

Aperiodic

events	Preconditions								Postconditions							
AE_1	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0
AE_2	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0
AE_3	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0
AE_4	0	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0
AE_5	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0
AE_6	0	1	1	0	0	0	0	0	1	0	0	0	0	0	0	0

Periodic

events	Postconditions							
PE_1	0	0	0	0	0	0	0	1
PE_2	0	0	0	0	0	0	1	0
PE_3	0	0	0	0	0	1	1	0
PE_4	0	0	0	0	1	0	0	0
PE_5	0	0	0	1	0	0	0	0

From the analysis of these matrices it can be shown that

Driven report:

AE_1 is driven by PE_1 and PE_2 or PE_3

AE_2 is driven by PE_2 or PE_3

AE_3 is driven by PE_3

AE_4 is driven by PE_4 and PE_5

AE_5 is driven by PE_5

AE_6 is driven by AE_2 and AE_1.

Drives report:

Periodic events:

PE_1 drives AE_1, AE_6 (known as path)

PE2 drives *AE1,AE6*

AE2,AE6

PE3 drives *AE3*, *PE4* drives *AE4*, *PE5* drives *AE5*

Aperiodic events: AE_1 drives AE_6 , AE_2 drives AE_6, AE_3 drives none, AE_4 drives none, AE_5 drivies none, AE_6 drives none.

The death of the conditions has not been considered, because the objective of this report is to observe the inter-dependence of the events. The inclusion of the limited life span for the conditions reduces this inter-dependence. Now, this report will vary with the user's requirement document. Any modification can be incorporated with corresponding changes in the precondition and postcondition matrices.

This basic concept to check logical correctness has been applied more systematically in an example SFS through execution path analysis technique as shown in Appendix B.2. The analysis of all execution paths for each periodic event allows the designer to check the logical correctness of the interaction of different periodic and aperiodic events to serve each sensing task. This attribute of this proposed framework allows the designer to avoid logical faults in the very early phase of system development.

2.6.2 Temporal Correctness

It has been explained that the *ith* periodic event should be served within $\min(\Delta Ts_i)$. This includes the time for PE_i and the execution times of subsequent dependent aperiodic events. Let the *ith* periodic event drives p paths. Each path can consist of any number of

aperiodic events for any number of times. Therefore, under this situation, the following condition should be satisfied

$$\min(\Delta Ts_i) \geq \max\{ Tp_{i1}, Tp_{i2}, ..., Tp_{ip}\} \tag{2.17}$$

Here, Tp_{ij} is the execution time for the *jth* path driven by the *ith* periodic event. The time required for the completion of a path can be calculated by the following relationship

$$\begin{bmatrix} Tp_{i1} \\ \cdot \\ \cdot \\ \cdot \\ Tp_{ip} \end{bmatrix} = \begin{bmatrix} a_{11} & \cdot & \cdot & \cdot & a_{1n} \\ \cdot & & & & \cdot \\ \cdot & & & & \cdot \\ \cdot & & & & \cdot \\ a_{p1} & \cdot & \cdot & \cdot & a_{pn} \end{bmatrix} \begin{bmatrix} AE_1.Te \\ \cdot \\ \cdot \\ \cdot \\ AE_n.Te \end{bmatrix} + \begin{bmatrix} PE_i.Te \\ \cdot \\ \cdot \\ \cdot \\ PE_i.Te \end{bmatrix} \tag{2.18}$$

Here, a_{ij} is the number of times the *jth* aperiodic event executes in path *i*. Therefore, from the user's data corresponding to the execution time of periodic and aperiodic requirements, the temporal correctness of the operation of the system can be verified.

$AE_i.Te$ is the maximum allowable time to accomplish the *ith* task. Now, within this constraint, Eq.(2.17) should be satisfied. Therefore, there is a need to optimize the distribution of execution times to the events. The relative complexity information of each task is necessary to derive a solution of this optimization problem. Let $AE_i.Pr.C$ be the complexity information of the process (computing component) to serve the *ith* event. The problem can be simplified by assuming that unit time is required to execute a single unit of complexity. In real life problem, the degree of complexity is usually variable. Therefore, the degree of complexity of the execution of every event has a lower and an upper bound. The upper bound is $AE_i.Te$ and the lower bound is greater than zero.

Now the optimization problem can be defined as

$$\textbf{Maximize } AE_i.te \text{ and } PE_i.te \tag{2.19}$$

Here, $AE_i.te$ is the execution time of the *ith* aperiodic event and $PE_i.te$ is the execution time of the *ith* periodic event.

Subject to constraints

$$\begin{array}{l} a_{10}PE_1.te + a_{11}AE_1.te + a_{12}AE_2.te + \cdots + a_{1n}AE_n.te \le \min(\Delta Ts_1) \\ \vdots \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \vdots \\ a_{m0}PE_m.te + a_{m1}AE_1.te + a_{m2}AE_2.te + \cdots + a_{mn}AE_m.te \le \min(\Delta Ts_m) \end{array} \tag{2.20}$$

$lower\ limit \le AE_i.te \ge AE_i.Te,\ lower\ limit \le PE_i.te \ge PE_i.Te$

Here, m is the total number of possible paths for all periodic events. If the *ith* periodic event contains P_i paths, $\min(\Delta Ts_i)$ will be copied to P_i rows. The total number of rows of this optimization matrix can be calculated by the following relation

$$m = \sum_{i=1}^{I} P_i \tag{2.21}$$

Here, I is the total number of periodic events (i.e., sensors). Using a mathematical programming technique, this optimization problem can be solved.

For an example sensor fusion system, every execution path for all seven periodic events corresponding to seven sensors has been analyzed as shown in Section B.2. Based on this analysis, the maximum service time for each periodic event has been estimated as shown in Table B.3. The distribution of the sensing times of all these seven sensors are shown in Fig. B.12. This analysis allows the developer to ensure temporal correctness for data acquisition from each sensor and the fusion of acquired. This capability of this proposed framework allows the developer to avoid temporal faults, resulting in improved system performance.

2.6.3 Reachability

The reachability problem is one of the most basic analysis problems associated with discrete event systems. Through this analysis it can be verified whether it is possible to fire an aperiodic event from a particular state of the system. The reachability set $R(DEVR, \mu)$ is the smallest set of states defined by,

if $\mu' \in R(DEVR, \mu)$ and $\mu''=\delta(\mu', E_j)$ for some $E_j \in E$, then $\mu'' \in R(DEVR, \mu)$.

Death of the conditions should be considered during the transition period of the system. The drives and driven report shown in the previous section will be the basis of this reasoning. From this analysis, the developer will be able to verify the possibility of execution of an event. Moreover, the developer will optimize the life-time of the conditions to ensure expected functionality of the system. At this stage, close co-operation between the developer and the customer will help ensure expected system functionality.

The reachability analysis of an example SFS is performed in section B.3. It is shown that using this proposed framework it is possible to perform reachability analysis of each periodic and aperiodic events.

2.6.4 Presence of Deadlock

A deadlock is a set of conditions such that every event which outputs to one of these conditions in the deadlock also inputs from one of these conditions. This means that once all of these conditions in the deadlock become consumed, the entire set of conditions will always be unmarked; no event can execute in the deadlock because there is no condition

available to fire an event. Under this condition, the system will go in an endless loop and will not be able to come back to the readiness state to serve the next periodic event generated by the clock. To overcome this problem, the developer can avoid recursive representation of the user's sensing requirements. Otherwise, special care must be taken so that all the conditions of the deadlock set are not consumed.

2.6.5 Repetitiveness

The repetitiveness is the condition that after the service of every periodic event, including the subsequent aperiodic events, the system will come back to the readiness state to serve the next periodic event. To ensure this, the last events of all the paths from all periodic events will generate only the readiness condition, certainly no other conditions. The left-most digit of the postcondition numbers of these events should be one and all other digits should be zero.

This basic concept to check repetitiveness has been applied successfully to verify that a sensor fusion system consisting of seven periodic events and eight aperiodic events is repetitive in its operation as shown in section B.3. This feature of this proposed framework allows the designer to avoid unwanted operating state (e.g., non-repetitive operating state) thus resulting in higher reliability of system operation. This knowledge component specifically contributes to the enhancement of the reliability of an intelligent system by avoiding such a catastrophic (system hung up) fault in the early stages of development.

2.7 The Utilization of the Operational Time

It has been explained previously that the system remains busy handling periodic and aperiodic events during certain intervals of time. The rest of the time the system is waiting in the readiness state to serve the next periodic event. The utilization factor of operation time is the ratio of the maximum busy period required to serve the events to the minimum waiting period in the readiness state. From the optimum distribution of execution times to both periodic and aperiodic events as proposed in section 2.6.2 it is possible to determine the maximum busy period of the system to serve the *ith* sensor by the following equation

$$Bp_i = PE_i.Te + \max\{Tp_{i1}, Tp_{i2}, \ldots, Tp_{io}\} \tag{2.22}$$

The graphical representation of the busy and idle periods of a typical system is shown in Fig. 2.14. The total busy period during the operational time t can be calculated by the following relation:

$$Bp(t) = \sum_{i=1}^{I} \frac{t - \Phi_i}{Ts_i} Bp_i \tag{2.23}$$

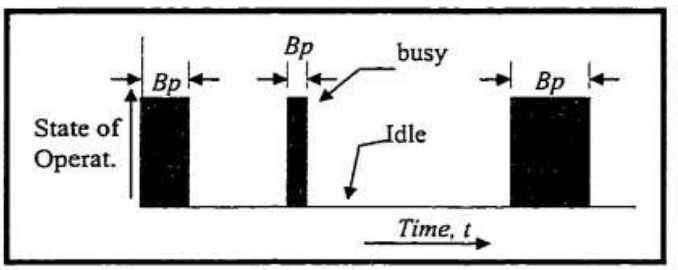


Figure 2.14.: The distribution of busy and idle periods in the DEVR model.

Therefore, the utilization factor of the operational time is

$$U(t) = \frac{Bp(t)}{t} = \frac{\sum_{i=1}^{I} \frac{t - \Phi_i}{Ts_i} Bp_i}{t} \tag{2.24}$$

The value of *U(t)* is also the measure of the utilization of the underlying computing system. By maximizing the use of the computational system, the developer can reduce the system cost. The increase of *U(t)* allows the developer to switch to the less powerful computing module for performing the same amount of computing job resulting in cost savings. Alternatively, this increase will allow the same computing module to incorporate more sensors. Therefore, it is desirable to increase the value of *U(t)*. The value of *U(t)* can be increased by, (1) increasing the value of Bp_i, and, (2) incorporating more sensors.

The utilization factor of the underlying computing system based on single processor to implement an example sensor fusion system (Fig.B.1) has been measured as shown in Eq.(B.47). The detailed specifications of this sensor fusion system is given in Appendix A. For this example system consisting of seven sensors, the utilization factor is 27.62%. This attribute of this proposed framework offers a quantitative means to measure system performance in terms of utilization of computing resources. This measure allows the justification of alternate sensing strategy as explained in sec. 2.7.1 and sec.2.7.2 for better utilization of the computing resources resulting in reduced system cost.

2.7.1 The Approaches to Increase the Value of the Busy Period

The sensors are grouped in a number of clusters. Each cluster is served by individual sensor fusion systems. One of the limiting factors for the maximum value of busy period, Bp_i, of the *ith* sensor in a cluster is $\min(\Delta Ts_i)$. This limiting factor is the function of the phases and the periods of all the sensors corresponding to that cluster. By changing the phases and periods of the sensors, the busy period can be increased as explained before (section 2.7). The value of Bp_i also can be increased by dropping sensors from a cluster or by reorganizing the sensors among the clusters.

2.7.2 Incorporation of More Sensors

The total busy period, $Bp(t)$, is the function of the total number of sensors when Bp_i is constant. Instead of $\min(\Delta Ts_i)$, if the maximum allowable execution period of the events limits the value of Bp_i, more sensors can be accommodated in the cluster in order to increase $Bp(t)$. The addition of more sensors to the cluster will make it more difficult to ensure $\min(\Delta Ts_i)$ for each sensor. Therefore, by selecting appropriate phases and periods of all the sensors more sensors can be added to increase $Bp(t)$ resulting in higher utilization factor.

2.8 Chapter Summary

The discrete event approach for modeling different modes of sensor data integration has been reported in this chapter. It is shown that different levels of sensor integration can be accomplished in this modeling framework. This is a generalized requirement modeling technique, which is not limited by the number of sensors. The number of sensors, which can be accommodated in a particular SFS, is limited by the sensing requirements. The quantitative basis of this attribute of this modeling paradigm is shown in Section 2.2. The technique to ensure logical and temporal correctness of a modeled SFS is demonstrated. A quantitative approach to measure the performance of a SFS in reference to the resource utilization is proposed in this chapter. The effectiveness of this proposed discrete event framework to model requirements of a multisensory sensing system is shown in Appendix A. It can be concluded that this novel quantitative system approach to model the requirements of sensor fusion system has enhanced the state-of-the-art of engineering methodology for developing the sensing sub-system of multisensory systems.

Chapter 3

Discrete Event Specifications of the Sensor Fusion System

3.1 Introduction

Generalized system theory allows the designer to decompose a discrete event dynamic system (DES) into components, which are also conceived as DES [60]. This hierarchical decomposition of the DES proceeds in cycles, each closer to the actual physical system. The component specification in *(n-1)* cycles becomes the system specification in the *nth* cycle. In the *nth* cycle, the system is again represented as dynamic interaction of components which are *(n+1)*th level systems. This hierarchical decomposition of the system complexity is shown in Fig. 3.1.

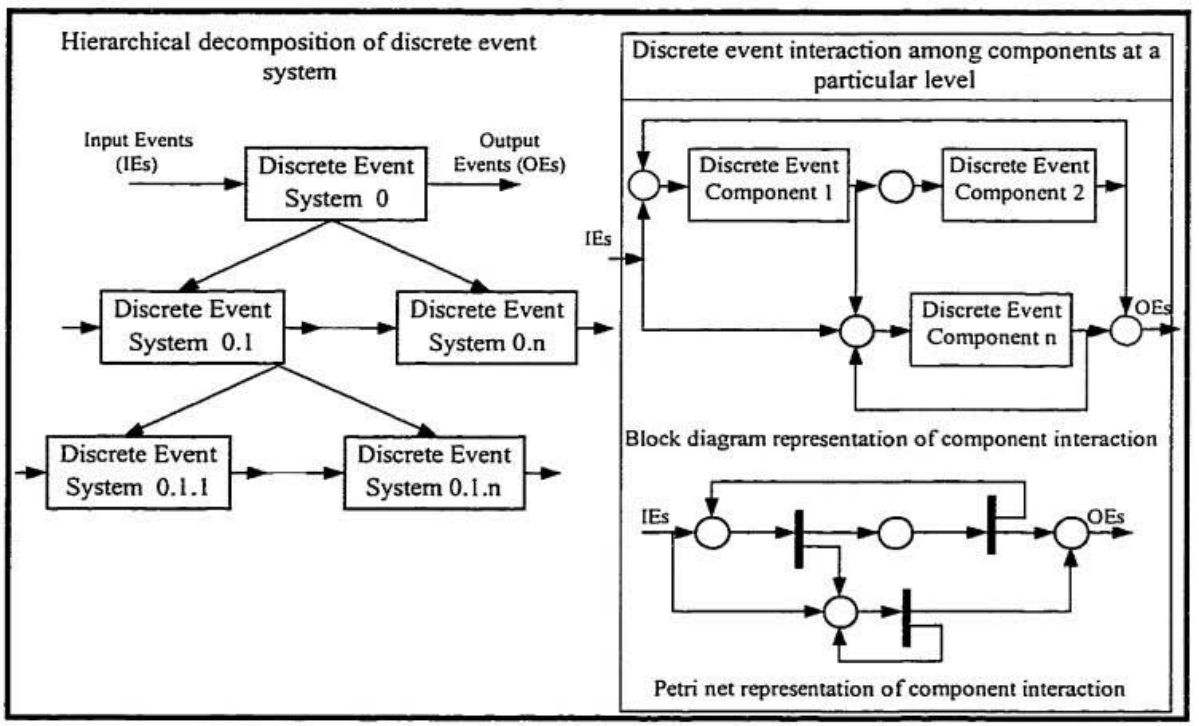


Figure 3.1: Hierarchical decomposition of a discrete event system.

It has been explained that sensor fusion system (SFS) is a discrete event dynamic system. Therefore, SFS is decomposable in hierarchical fashion using generalized system theory. The discrete event requirements model is the zero level representation (i.e., system requirements from user's perspective). In this level, the requirement engineer interacts with the customer to develop the virtual prototype of the system. This prototype defines the user's discrete requirements and their dynamic interaction. The preconditions, postconditions and execution time of each discrete requirement as shown in Fig. 3.2 are defined in the requirement modeling stage. These discrete requirements will be conceived as systems. These systems will be decomposed as dynamic interaction of next level sub-systems. This process of recursive decomposition will continue to reach the level to represent the system as discrete event dynamic interaction of discrete computing components. The number of level of decomposition depends upon the complexity of the system as well as the size of the computing components.

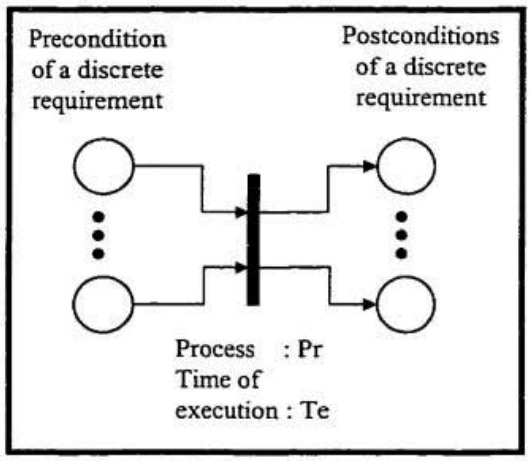


Figure 3.2: Petri net model of a discrete requirement.

In each level, a system (micro system) will be defined as preconditions, postconditions and execution time. Each micro system is connected to the rest of the system through the interface of preconditions and postconditions. Here, preconditions and postconditions are the input and output of the system according to the conventional system theory. The dynamic interaction of the constituting components must satisfy the functional and temporal specification of these micro systems. It has been explained that

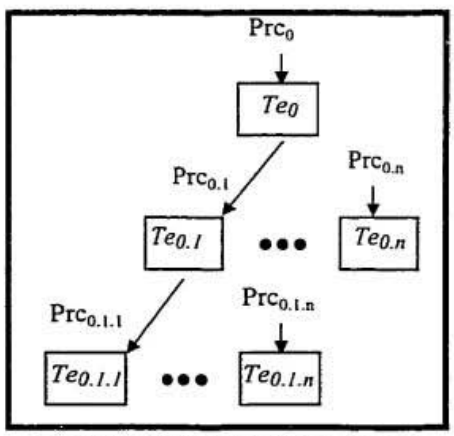


Figure 3.3: Flow of functional and temporal specifications.

components in the *(n-1)th* level are treated as systems in the *nth* level of decomposition. Therefore, functional and temporal specifications flow from higher level, *(n-1)*, to lower level, *n*, as shown in Fig. 3.3. The functional and temporal performance of the *(n-1)th* level must be supported by the dynamic interaction of the components in the *nth* level. This is the job of the discrete event specification (DEVS) model to establish the link between the *(n-1)th* level and *nth* level. This hierarchical decomposition will also allow the simulation of the discrete event model in modular fashion to cope with the complexity of the Petri net model, which usually grows exponential with the increase of system size.

3.2 Discrete Event Dynamic Interaction of the Computing Components in the SFS

A system in the *nth* level (i.e., a component in the *(n-1)th* level) is realized by the sequence of dynamic interactions of the components. In an SFS, these sequences of interactions are conditional. In order to satisfy the functional and temporal specifications of the system, the execution of all of these traces should be accomplished within the defined time window, $T_{n.i}$, the execution time of the *ith* system at the *nth* level determined by the *(n-1)th* level specifications.

3.2.1 Functional Specification of the System

The interaction among the components in a particular sequence (trace) guarantees the functional specification of the system. A trace may be elementary or compound. In an elementary trace, all the components are connected one after another forming a series as shown in Fig. 3.4(a) [1],[32]-[35]. This elementary trace consists of sequential operation of four components as represented by the Eq. 3.1.

$$Se = \{c_1, c_2, c_3, c_4\} \tag{3.1}$$

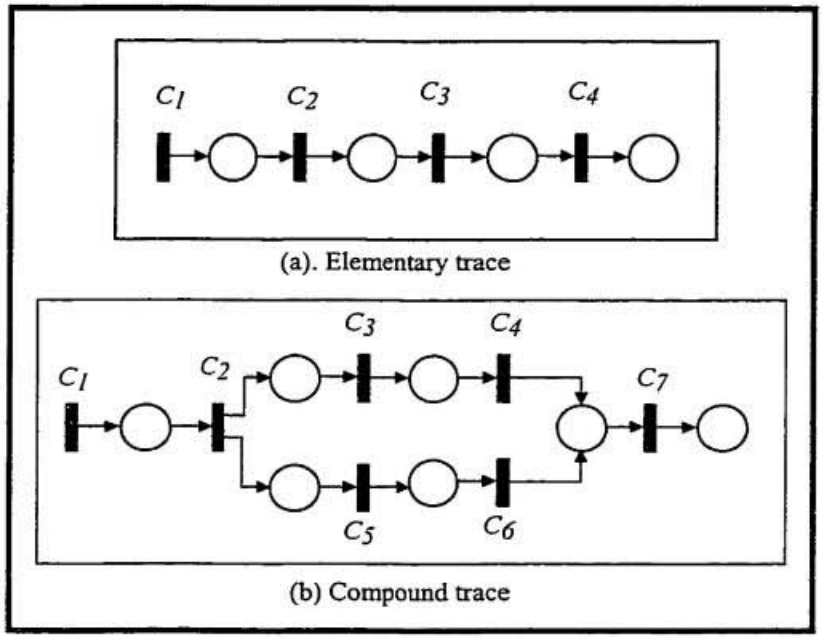


Figure 3.4: Examples of elementary and compound traces.

A compound trace consists of a number of elementary traces connected in series and/or parallel pattern as shown in Fig. 3.4(b). Mathematically, this trace can be represented by the following equation

$$Sc = \{Se_1, Sc_1, Se_2\} \tag{3.2}$$

Here, $Se_1 = \{c_1, c_2\}; Sc_1 = \{Se_{11}, Se_{12}\}; Se_{11} = \{c_5, c_6\};$

$Se_{12} = \{c_3, c_4\}; Se_2 = \{c_7\}$

The systematic approach for the formation of these traces is explained in the following sub-sections.

3.2.1.1 Formation of Elementary Traces

Let the component set consists of n number of components, $\boldsymbol{C}=\{c_n: n \in \boldsymbol{Z}\}$. The formation of m elementary traces can be represented by the following matrix relation:

$$\begin{bmatrix} Se_1 \\ Se_2 \\ . \\ . \\ . \\ Se_m \end{bmatrix} = \begin{bmatrix} P_{11} & P_{12} & . & . & . & P_{1n} \\ P_{21} & P_{22} & . & . & . & P_{2n} \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ P_{m1} & P_{m2} & . & . & . & P_{mn} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ . \\ . \\ . \\ c_n \end{bmatrix} \tag{3.3}$$

$$\boldsymbol{Se=P\,C} \tag{3.4}$$

Here, $\boldsymbol{Se}$, $\boldsymbol{P}$ and $\boldsymbol{C}$ are the trace, position and component vectors respectively. The values of the elements of $\boldsymbol{P}$ indicate the position of a component in the corresponding trace; the null value stands for the absence of the component. The following example explains this trace formation phenomena:

$$\begin{bmatrix} Se_1 \\ Se_2 \\ Se_3 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 1 \\ 3 & 1 & 2 \\ 0 & 2 & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} \tag{3.5}$$

The elements of the first row of the position matrix defines that the component c_1 has second position ('2'), component c_2 does not participate ('0') and the third component c_3 is in first position in the formation of the elementary trace Se_1. The matrix relationship shown in Eq.(3.5) forms the following traces:

$$Se_1 = \{c_3, c_1\} \tag{3.6}$$

$$Se_2 = \{c_2, c_3, c_1\} \tag{3.7}$$

$$Se_3 = \{c_3, c_2\} \tag{3.8}$$

In this matrix operation, '+' operator of conventional matrix algebra has been replaced by '***position operator***', which uses the elements of ***P*** matrix to position the corresponding components of ***C*** to form a trace.

3.2.1.2 Formation of Compound Traces

A compound trace is composed of a number of segments, $Sg=\{Sg_i: i \in Z\}$. Each segment may form as

- ***a trace*** (elementary or compound);
- ***a parallel operation*** of more than one elementary and/or compound traces;
- ***a branch*** to other traces (elementary and/or compound);
- ***a loop*** of a trace (elementary and/or compound) ;

Segment formation operators (So):

st $\cong$ ***(1,p)*** : pick one trace from a set of traces

pr $\cong$***(2,p)*** : runs more than one trace in parallel

br $\cong$***(3,p)*** : select one trace from a set of traces to branch

lp $\cong$***(4 ,p)***: runs a trace multiple number of times.

Here, ***p*** stands for position information.

Segments are of two types:

- **Elementary segment**, $Sge_i: i \in Z$; composed of only elementary traces related by segment formation operators.

- **Compound segment**, $Sgc_i: i \in Z$; the constituting elements are compound traces related by segment formation operators.

Formation of simple segments can be defined by the following matrix relationship:

$$\begin{bmatrix} Sge_1 \\ Sge_2 \\ . \\ . \\ . \\ Sge_i \end{bmatrix} = \begin{bmatrix} So_{11} & So_{12} & \cdot & \cdot & \cdot & So_{1j} \\ So_{21} & So_{22} & \cdot & \cdot & \cdot & So_{2j} \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ So_{i1} & So_{i2} & \cdot & \cdot & \cdot & So_{ij} \end{bmatrix} \begin{bmatrix} Se_1 \\ Se_2 \\ . \\ . \\ . \\ Se_j \end{bmatrix} \tag{3.9}$$

An example to explain the above segment formation relationship is as follows:

$$\begin{bmatrix} Sge_1 \\ Sge_2 \end{bmatrix} = \begin{bmatrix} (2,1) & (2,2) \\ (3,2) & (3,1) \end{bmatrix} \begin{bmatrix} Se_1 \\ Se_2 \end{bmatrix} \tag{3.10}$$

The first row of segment formation operator matrix defines that the first elementary segment Sge_1 is formed by the parallel operation of elementary traces Se_1 and Se_2. The second elementary segment is formed by the branching operation of the elementary traces Se_1 and Se_2 defined by the elements of the second row of the segment formation operator matrix. Therefore, newly formed elementary segments can be defined by the following equations:

$$Sge_1 = \mathbf{pr}\{Se_1, Se_2\} \tag{3.11}$$

$$Sge_2 = \mathbf{br}\{Se_2, Se_1\} \tag{3.12}$$

The Petri nets models of these two traces are shown in Fig. 3.5.

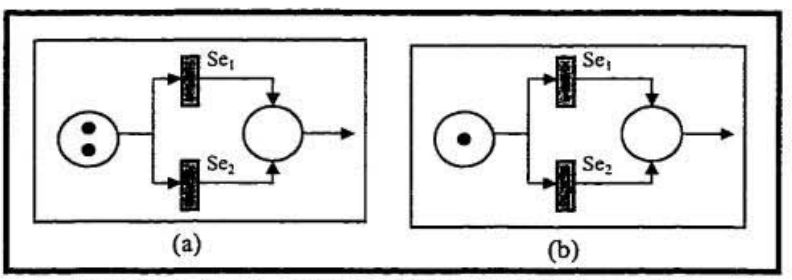


Figure 3.5. Formation of elementary segments.

The compound traces can be formed by the following matrix relationship:

$$\begin{bmatrix} Sc_1 \\ Sc_2 \\ . \\ . \\ . \\ Sc_m \end{bmatrix} = \begin{bmatrix} P_{11} & P_{12} & . & . & . & P_{1n} \\ P_{21} & P_{22} & . & . & . & Pn \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ P_{m1} & P_{m2} & . & . & . & P_{mn} \end{bmatrix} \begin{bmatrix} Sg_1 \\ Sg_2 \\ . \\ . \\ . \\ Sg_n \end{bmatrix} \tag{3.13}$$

In the first iteration, segments, Sg_i, are simple, but in the following iterations they are compound. The following matrix relation defines the formation of compound segments.

$$\begin{bmatrix} Sgc_1 \\ Sgc_2 \\ . \\ . \\ . \\ Sgc_i \end{bmatrix} = \begin{bmatrix} So_{11} & So_{12} & . & . & . & So_{1j} \\ So_{21} & So_{22} & . & . & . & So_{2j} \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ So_{i1} & So_{i2} & . & . & . & So_{ij} \end{bmatrix} \begin{bmatrix} Sc_1 \\ Sc_2 \\ . \\ . \\ . \\ Sc_j \end{bmatrix} \tag{3.14}$$

The formation of compound trace as well as compound segments is recursive in nature. These trace formation processes using matrix operations enable the developer to specify the component interaction through the values of elements of the matrix. This quantitative representation of the dynamic interaction among the components facilitates the development of design automation software. It should be noted that the use of recursive for stepwise reduction of system complexity is different from recursive representation of user requirements. The recursive representation of user requirements has the potential to create deadlock in system operation.

The representation of the processes to serve the aperiodic events of an example sensor fusion system through elementary and compound traces is shown in section C.2.1.

3.2.2 Verification of the Logical Correctness of the Functional Specifications

The methodology to check the logical correctness of the discrete event requirement model as explained in chapter 2 can be used to verify the logical correctness of the executions of the components to realize the system specification. Moreover, other related issues, as such avoidance of deadlock, reachability, and repetitiveness in the dynamic interaction of the components can be checked in the similar way.

The decomposition of the processes to serve eight aperiodic events in terms of interaction among sixteen computing components of an example sensor fusion system has been shown in section C.2.1. This decomposition allows the verification of the logical correctness of the component interaction in the similar way as shown for the DEVR model. This is one of the most important advantages of using the same modeling technique at different phases of the development cycle.

3.3 Determination of the Temporal Specifications of the Computing Components of the Sensor Fusion System

It should be noted that the execution of an event is conditional in an SFS. The generations of postconditions for the executions of the processes are variable. Due to the firing of a process, a fraction of the total conditions,

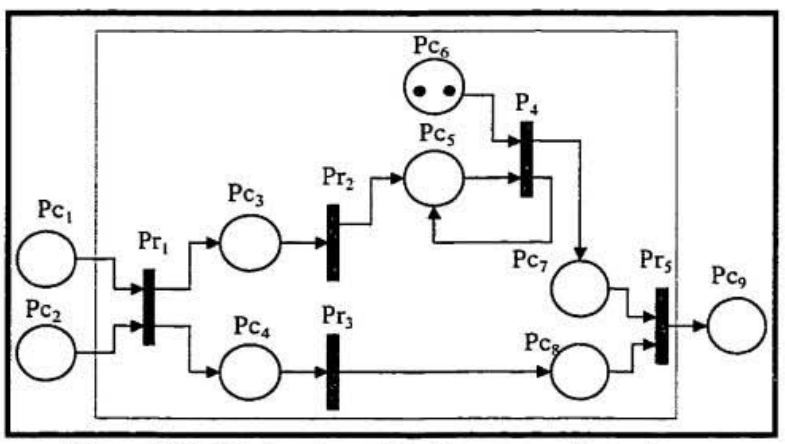


Figure 3.6: Petri net model of an example system.

no condition or all of the conditions, may be generated. This situation can be explained by an example shown in Fig. 3.6. The possible firings of the processes are summarized in Table 3.1 (this is not the exhaustive list). Now, for any sequence of firing of the processes, the total execution time of any sequence or trace should be smaller than or equal to the system response time. The system includes of a number of parallel traces, and the system's temporal specification should be greater than the temporal longest trace.

Table 3.1: The possible firing sequences of the processes of an example SFS as shown in Fig. 3.6

Paths	Firing of processes				
	Pr_1	Pr_2	Pr_3	Pr_4	Pr_5
1	0	0	0	0	0
2	1	0	0	0	0
3	1	1	0	0	0
4	1	1	0	1	0
5	1	1	0	2	0
6	1	1	1	2	1

The execution of the system can be abstracted as the execution of m parallel traces, S= $\{S_i,\ 1 \leq i \geq m\}$ as shown in Fig. 3.7. It should be noted that the executions of these traces are conditional as explained (section 3.1). In the best case, no trace may be required to execute; but in the worst case all the traces may be required to execute. The design should be based on the worst case situation, and the executions of all these traces should satisfy the following relation:

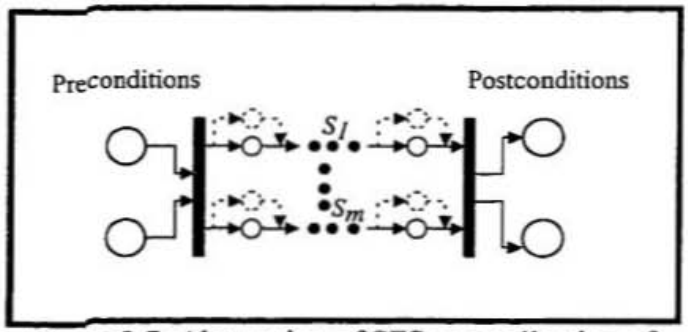


Figure 3.7: Abstraction of SFS as a collection of parallel traces.

$$T_{n,i} \geq \max\{St_1, St_2 \ldots St_m\} \tag{3.15}$$

Here, St_i is the execution time of the *ith* trace.

This proposed technique is applied successfully to compute the service time of eight aperiodic events for an example sensor fusion system as shown in section C.2. These aperiodic events have been served by the interaction of a set of 16 components.

The execution times required for m traces for n components having execution time c_{ti} can be calculated using the following matrix relation

$$\begin{bmatrix} St_1 \\ \cdot \\ \cdot \\ \cdot \\ St_m \end{bmatrix} = \begin{bmatrix} a_{11} & \cdot & \cdot & \cdot & a_{1n} \\ \cdot & & & & \cdot \\ \cdot & & & & \cdot \\ \cdot & & & & \cdot \\ a_{m1} & \cdot & \cdot & \cdot & a_{mn} \end{bmatrix} \begin{bmatrix} c_{t1} \\ c_{t2} \\ \\ \\ c_{tn} \end{bmatrix} \tag{3.16}$$

The coefficient a_{ij} is the number of times the *jth* component is present in the formation of the *ith* trace. A zero value of this coefficient indicates that the *jth* component is not part of the formation of the *ith* trace. Usually there is a domain of selection of component execution time, and there is a scope for optimization in the distribution of time to the components. The objective of this optimization is to allow maximum component execution time. There is a cost to lowering the execution time of the components. This cost may be due to the requirement of additional research to lower the computational complexity of the components, or to use computationally less powerful algorithms to do the same type of work sacrificing quality, or to switch to higher cost processor. The objective of optimization is to maximize the execution time of each component while satisfying the temporal specification of each trace.

Maximize c_{ti} ; **Subject to**

$$\begin{array}{c} a_{11}.c_{t1} + a_{12}.c_{t2} + a_{13}.c_{t3} + \cdots + a_{1n}.c_{tn} \le T_{n,i} \\ \cdot \\ \cdot \\ \cdot \\ a_{11}.c_{t1} + a_{12}.c_{t2} + a_{13}.c_{t3} + \cdots + a_{1n}.c_{tn} \le T_{n,i} \end{array} \tag{3.17}$$

and

$$c_{t_l,i} \le c_{ti} \le c_{t_h,i} \tag{3.18}$$

Here, $c_{t_l,i}$ is the lower limit and $c_{t_h,i}$ is the upper limit of the *ith* component execution time.

This optimization function has been successfully utilized to select the execution times of a set of computing components to serve the aperiodic events of an example sensor fusion system as shown in section C.3. The application of a formal quantitative approach to optimize the use of system resources is one of the novelties of this research work. This knowledge component enhances the state-of-the-art of the practice of scientific methods to engineer sensor fusion systems.

3.3.1 Sensitivity Analysis of the Components Execution Times

The sensitivity analysis of the component execution times is a measure of the rate of change of the total distribution of execution time of the components due to the change of execution time of a particular component. Due to a change Δt_i amount in execution time of the *ith* component, the total change of the execution of time, ΔT, over n-1 components can be calculated by the following relation:

$$\Delta T = \sum_{\substack{j=1 \\ j \neq i}}^{n} \Delta t_j \tag{3.19}$$

Therefore, the sensitivity of the execution time of the *ith* component is

$$Sen_i = \frac{\Delta t_i}{\Delta T} \tag{3.20}$$

This sensitivity analysis can be related to cost optimization. If C_i is the cost related to the per unit change of computational time for the *ith* component, the cost sensitivity of the *ith* component can be measured by the following relation

$$Csen_i = \frac{C_i \Delta t_i}{\sum_{\substack{j=1 \\ j \neq i}}^{n} C_j \Delta t_j} \tag{3.21}$$

An investment to decrease the computational time of the *ith* component is justified as long as the value of $Csen_i$ is greater than one. Therefore, this basic relationship can be used to justify the investment on the research to develop computationally less complex components. Moreover, this finding can be used to justify the decision to implement highly sensitive computing component in dedicated hardware or in other special type of devices to reduce the overall system cost.

The sensitivity analysis of the component execution times will enable the developer to optimize the system cost, while maintaining temporal performance of the system. Similar analysis can be performed on the discrete event requirement model to determine the rate of change of the optimum service times of both periodic and aperiodic events. This development has added scientific knowledge in the engineering process of such system development.

3.4 The Reliability Aspects of the Discrete Events Specification

The DEVS model generates the functional and temporal specification of each computing component and represents the dynamic interaction among these components. This model satisfies the execution of each discrete requirement as defined in the discrete event requirement model. The Petri net based modeling tool as proposed in this chapter has enough power to ensure the realization of the functional and temporal specifications of each discrete requirement by defining the specification of each component and the dynamic interaction among them. Therefore, it can be concluded that this model has enough potential to avoid error in mapping the user's sensing requirements to the component level specification. This thesis has addressed the problem of enhancement of reliability of SFS by avoiding temporal and logical faults in the development phase. This objective can be partially achieved by practicing this proposed engineering methodology as explained in this chapter.

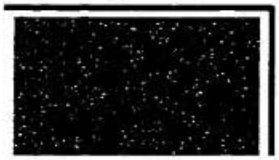
3.5 The Modeling of Multi-node based Sensor Fusion System

In a multi-node based SFS (e.g., distributed sensing system of mobile robots, distributed air defense system), the nodes are organized in hierarchical fashion [6]. In this hierarchical organization, each leaf node consists of processing unit and a set of associated sensors. At each level, nodes receive information from lower-level nodes, integrate the information received according to their position in the hierarchy, and send extracted information to nodes at next higher level. This proposed discrete event based modeling technique can be directly applied to model leaf level sensing nodes. To model higher level nodes, associated each lower level node can be abstracted as virtual sensor. Abstracting each lower level node as virtual sensor this modeling approach can be applied to model each node of different levels. Therefore, the discrete event modeling approach reported in this thesis can be used to model multi-node based sensor fusion system as well.

3.6 Chapter Summary

The hierarchical decomposition of DEVR model to component level specification has been proposed. A mathematical framework to model the interaction of the components in terms of trace and segments has been shown in section 3.2. This quantitative abstraction of component interaction enables the automation of the design work. The proposed technique (section 3.3) to optimize the component execution times provides means for improved utilization of system resources. This novel technique has been used successfully to derive component level specification from the DEVR model of the example SFS shown in Appendix B. This proposed formal method of mapping the user's sensing requirements to component level specifications is an important contribution of this thesis.

Chapter 4



The Architecture of the Embedded Computing System to Implement the DEVS Model of the SFS

4.1 Introduction

The discrete event requirements model (DEVR) of a sensor fusion system (SFS) defines the interaction of user level events and allocates optimum times to the execution of the corresponding processes. The discrete event specifications model (DEVS) defines the computing component level interaction to implement the DEVR model of the system under development. The underlying computing system must execute every computing component within the time window as specified by the DEVS model. The architecture of the computing system should be optimized to economize the cost as well. The work load offered to the system is defined by the DEVR model. Therefore, the architecture should be reasoned from the DEVR model.

4.2 The Execution Time of a Computing Component

The interaction of the computing components is defined in the DEVS model. The execution of these components must be accomplished by the redefined time windows derived from the DEVR model. In terms of computational complexity, a component can be represented by a five-tuple vector as shown in the following equation

$$\boldsymbol{C} = \{I, F, D, M, H\} \tag{4.1}$$

Here, *I, F, D,* and *M* stand for computation load of integer, floating point, compare and memory operations respectively. The fifth tuple *H,* represents other execution overhead (e.g., component specific system management job). Due to the presence of decision making and looping operations in a computing component, estimates for plausible values for the tuples are usually subjective. But in a SFS system, the components should be executed within defined time windows. Therefore, the temporal worst case operation scenario should be considered when calculating the values of the tuples.

The execution time of a computing component on a particular processor can be calculated by the following equation

$$c_t = \frac{1}{f} (P_I I + P_F F + P_D D + P_M M + P_H H) \tag{4.2}$$

Here, P_I, P_F, P_D, P_M, and P_H represent the estimated required clock pulses to execute each unit of integer, floating point, compare, memory operation and execution overhead operations respectively. The time period of the processor clock is represented by $1/f$. The execution time of simple and compound traces as explained in the DEVS model in the previous chapter (Sec. 3.2.1) can be calculated by the equations

$$Se_t = c_{t1} + c_{t2} + c_{t3} + c_{t4} \tag{4.3}$$

$$Sc_t = Se_{t1} + Sc_{t1} + Se_{t2} \tag{4.4}$$

$$Sc_{t1} = max(Se_{t11}, Se_{t12}), \text{ for parallel operation} \tag{4.5}$$

$$Sc_{t1} = Se_{t11} + Se_{t12}, \text{ for sequential operation} \tag{4.6}$$

4.3 The Reasoning Basis of the Architecture of the Computing System

The quantitative relations developed so far help us to reason about the architecture of the embedded computing system. The request for the execution of a computing component by the computing system can be considered as a discrete event. The arrival and service of these events can be modeled as single server queue as shown in Fig. 4.1. Now to serve an event properly the following condition must be satisfied

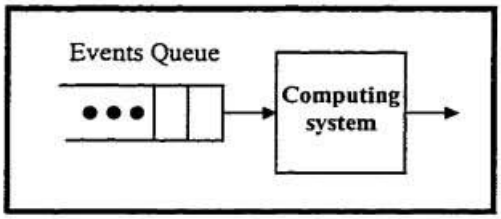


Figure 4.1 A model of embedded events service system

$$Wq + Ts \leq Te \tag{4.7}$$

Here, Wq is the expected waiting time of an event in the queue; Ts is the execution time of the corresponding process; Te is the time window within which the event must be served to satisfy the DEVS model. To simplify the modeling process, the arrival of the events (i.e., the request to execute the computing components) is considered as a Poisson process [61]. In this simplified representation of the problem, the request of the execution of only one type of computing component arrives in the queue in a random fashion. Then the expected waiting time in the queue can be shown as [61]

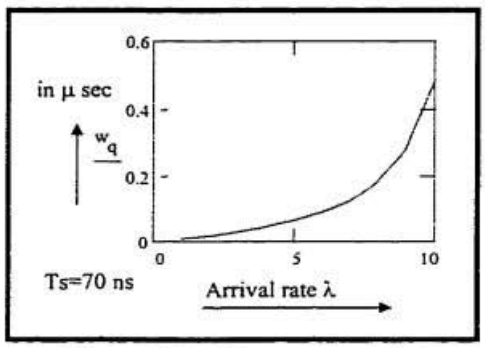


Figure 4.2: The exponential growth of the waiting time with the increase of the arrival rate.

$$W_q = \frac{\lambda}{\frac{1}{Ts}(\frac{1}{Ts} - \lambda)} \tag{4.8}$$

Here, λ is the arrival rate of the events in the queue of the computing system. The waiting time increases exponentially with the increase of arrival rate as shown in Fig. 4.2. Here, to simplify the problem the service time to each event is assumed to be the same. For a particular event, the service time *Te* is predetermined from the DEVS model. In the DEVS model, *Te* represents the component execution time.

4.4 The Architecture of the Computing System to Execute Elementary Traces

In an elementary trace as explained in the previous chapter (sec.3.2.1), the execution of the components takes place in a sequential manner. The outputs of the upstream components become the inputs of the downstream components. The existence of parallelism in the execution of these components can be detected using Bernstein's conditions[62]. The Bernstein's conditions state that two processes P_1 and P_2 with their input sets I_1 and I_2 and output sets O_1 and O_2, respectively can execute in parallel if the following conditions are satisfied:

$$I_1 \cap O_2 = \phi. \text{ Here, } \phi \text{ is the empty set} \tag{4.9}$$

$$I_2 \cap O_1 = \phi \tag{4.10}$$

$$O_1 \cap O_2 = \phi \tag{4.11}$$

Therefore, from these conditions it is evident that the components in the elementary traces cannot be executed in parallel. It should also be noted that the executions of upstream components generate the events to execute downstream components. Under this circumstance, no event waits in the queue to be served resulting in zero waiting time. Therefore, the execution time of the components composing the elementary traces should be equal to or less than the time windows specified in the DEVS model. The speed of the execution of a elementary trace is limited by the computing power of a single processing

units. Under this operational scenario, a single node based computing system is a solution for executing elementary traces. The developer can negotiate with the customer to redefine the time requirements to increase the time windows of the elementary traces to switch to a less powerful processor.

4.5 The Architecture of the Computing System to Execute Compound Traces

A compound trace consists of a number of elementary traces connected in sequential and in parallel manner as explained in the previous chapter (sec. 3.2.1.2). The sequential part does not satisfy the Bernstein's conditions as explained in the previous section 4.4. But the parallel part satisfies those conditions. Therefore, a multinode architecture can be used to exploit the power of commodity microprocessors to realize a cost effective sensor fusion system as shown in Fig. 4.3. The parallel elementary traces can be executed in sequential fashion as well, as shown in Eq. (4.5, 4.6). In that case the waiting time will be equal to the execution of the previous parallel elementary traces. Therefore, both single node and multiple nodes based architecture can be used to implement compound traces. In the distribution of time windows to the events in the DEVR and DEVS models the parallelism of the service of the events must be considered for optimum allocation. This makes it very clear that the DEVR and DEVS models are directly related to the architecture of the underlying embedded computing system. Therefore, both DEVR and DEVS model, and the

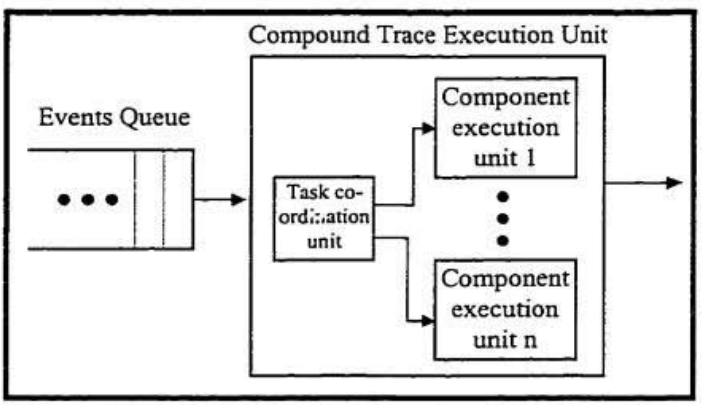


Figure 4.3: Multiple nodes based architecture of the embedded computing system to execute compound traces.

architecture of the computing system should be recursively optimized to reach an optimum solution. In some cases, there may exist parallelism in the DEVR and DEVS model but due to the cost consideration the underlying computer system may adopt single node based architecture. It should be noted that multiple nodes based architectures can enhance the execution speed only if there exists parallelism in the DEVR and DEVS model. Therefore, a Petri net based formalism in the modeling of the requirements and specification will enable the designer to determine the optimum architecture of the embedded computing system. This is one of the novelties of this work to link the different stages of the development process through a uniform reasoning technique. The higher level abstraction in the early cycles of the development process gradually proceeds to physical system in the latter cycles through sound quantitative reasoning. This reasoning basis has been utilized to reason the basic structure of the embedded computing system of the example sensor fusion system as explained in section D.2 and section D.3.

4.6 Implementation of Multiple DEVS Models on a Single Computing System

The method proposed in section 2.2 to avoid overlapping of sensing periods in order to satisfy periodic requirements may not keep the underlying computing system busy during the entire period of operation. Rather the system will remain busy for a defined period of time as determined by the DEVR model as shown in Fig. 2.11. Therefore, the

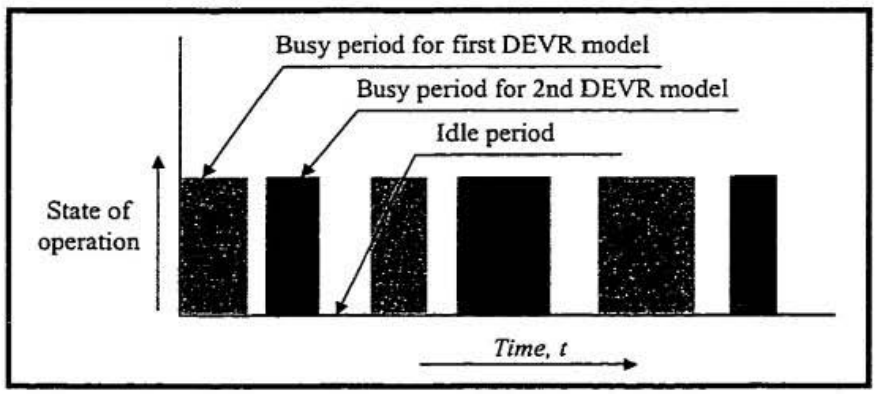


Figure 4.4. Interlacing of two DEVR models to increase the utilization factor of the embedded computing system.

multiple DEVR model can be interlaced to be implemented on a single computing system. The implementation of two DEVR models in interlaced fashion is shown in Fig. 4.4. This will increase the total busy period of the system resulting in higher utilization factor of the underlying computing system. To utilize this interlacing concept, sensors can be divided into multiple clusters and each cluster will have its own DEVR model. This will increase design complexity, because these different DEVR models should be interlaceable. The discrete event formalism proposed in this thesis will help deal with this complexity transparently. There is a scope to increase the utilization factor of the computing system interlacing DEVR models of different SFSs resulting in cost effective solution. This is a new concept about the architecture of a sensor fusion system.

4.7 Randomness of Execution Times of Computing Components on Modern Processors

It has been mentioned (Sec.4.2) that the computational time of a computing component is a function of a number of variables as defined by Eq. 4.1. The DEVS model allocates definite time windows to the components. In order to satisfy the DEVS model, the component execution time must be predictable and the maximum value must be within the corresponding window.

Over the years the architectures of microprocessors have adopted pipeline architectures to achieve high average throughput. This pipeline feature, however, has incorporated randomness in the actual execution time of a computing component as explained in sec. D.5. The actual execution time not only depends upon the computational complexity of the component, but also upon the sequence of instructions constituting the component. The reasons behind this randomness are the hazards that prevent the next instruction in the instruction stream from executing during its designated clock cycle [63]. These

hazards include structural hazard, data hazard and control hazard [63]. The occurrence of these hazards will stall the pipeline and the execution time of an instruction will be extended by an undefined amount depending upon the nature of the other previous instructions in the stream. Therefore, the value of the tuples *I*, *F* and *D* should be computed by running the component on the actual processor. The value computed by analyzing the computational complexity will not provide the basis of calculating actual execution time. If this is not done in the prescribed way, it will be very difficult or may be impossible to satisfy the DEVS model in the implementation phase in a cost-effective manner. As a result the reliability of the system will suffer.

The statistics of clock cycles per instruction (CPI) for different benchmark programs running on a modern processor has been reported in Section D.3 [63]. From these statistics, it appears that the variations of CPI in these programs are more than 100%. Therefore, the consideration of this source of randomness in the component execution time is very important in estimating the actual computing time.

Hierarchical memory organization has been accepted as a realistic approach in building memory system [64]. Memory devices at a lower level are faster to access, smaller in size, and more expensive per byte, having a higher bandwidth and using a smaller unit of transfer as compared with those at higher level. The effective access time of a data unit from the memory can be defined by the following equation [62]

$$T_{eff} = \sum_{i=1}^{n} f_i . t_i \tag{4.12}$$

$$T_{eff} = h_1 t_1 + (1-h) h_2 t_2 + (1-h_1)(1-h_2) h_3 t_3 + \cdots + (1-h_1)(1-h_2)\cdots(1-h_{n-1}) t_n \tag{4.13}$$

Here, f_i is the access frequency (probabilistic term) to *ith* level, t_i is the access time to *ith* level, h_i is the hit ratio (probabilistic term) at *ith* level. Therefore, it is evident from these equations that data access time is random, and depends upon the distribution of data in different memory levels. The processors developed in recent past (e.g., Pentium, Alpha)

have internal memory known as on-chip cache [63]. Therefore, the designer of the embedded computing system to implement the DEVS model has little room to avoid this randomness in data access time. It should be noted that this randomness not only depends upon the data access pattern within the computing component, but also upon the initial data distribution at the different levels of memory due to data access of the previous computing components. It has been mentioned that the randomness of the generation of aperiodic events will make it virtually impossible to predict the distribution of data at different memory levels. Therefore, it is recommended that the data distribution in the different levels of the memory system must be initialized to obtain predictable value of the *4th* tuple, *M*. Due to the requirement of this initialization at the beginning of the execution of each computing component, the value of the *5th* tuple *H* will increase. But but this will help to make good estimate of the execution time of a particular component resulting in higher reliability of the implementation of the DEVS model. This problem can be dealt with as well by measuring worst case value if the system requirements do not put constraints on lower bound of execution times.

A quantitative example of different scenarios of data distributions at different memory levels has been shown in Table D.6. From this example problem, it appears that the randomness of initial data distribution may make the actual execution times of the components highly random. The consideration of this source of error is important to ensure temporal correctness of the DEVS model in the operational stage.

4.8 Chapter Summary

The rationale of selecting the architecture of the embedded computing system for implementing the DEVR model of a SFS has been outlined here. The hazards to ensure temporal correctness of the DEVR model in the implementation phase have been identified and potential solutions to address these hazards have been proposed. The quantitative framework discussed in this chapter will help the developer implement a sensor fusion system ensuring temporal correctness in a cost-effective manner.

Chapter 5

Hardware Fault-Tolerance of the Sensor Fusion System (SFS)

5.1 Introduction

From the hardware perspective, an SFS is a physical system consisting of a number of electronic components and sensors connected in a particular fashion. Mechanical components (e.g., connectors, wires, PCB) to support the system are not part of this work. Due to the effect of the manufacturing process, aging, and operating conditions these components may fail to do their job in course of time. Therefore, the reliability of the system is a function of the mean time to failure (MTTF) of these components [65]. Due to the requirement of the unsupervised operation of the SFS for a prolonged period of time in safety and mission critical operations, the failure of these components should be detected and a faulty component should be replaced by a fault-free one autonomously. The detection and replacement of the faulty components will be treated as if these components did not fail, provided that this fault clearance operation is not detrimental to the system's performance. This redundancy in the use of components has the potential to increase the reliability of the system [57]. This is based on the argument that if there are n identical components in parallel, on the

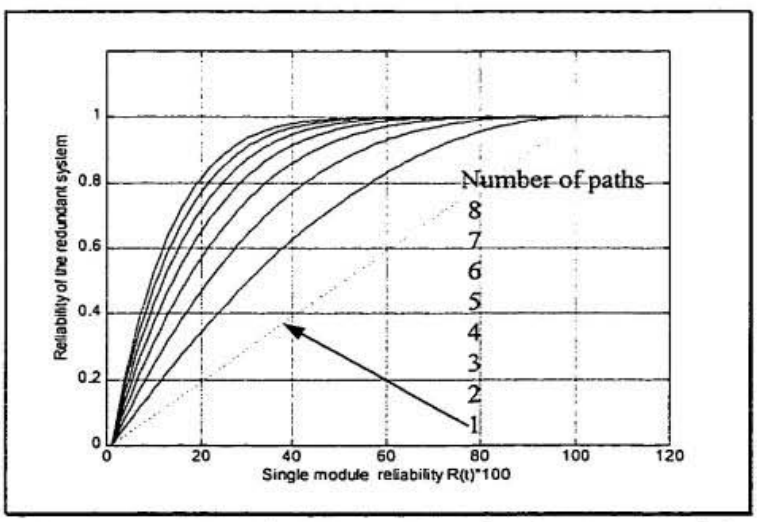


Figure 5.1: The reliability profile of redundant parallel system.

assumption that only one working component will provide the required output, the reliability of this parallel system is

$$R = 1 - p^n \tag{5.1}$$

Here, p is the probability of failure of each parallel component. The reliability profile of such parallel system for different numbers of parallel sensors (paths) is shown in Fig. 5.1.

Due to the need of some mechanism for checking the working component and switching to the next component when it fails, a better figure for overall reliability is given by multiplying R by the reliability of the checking and switching mechanism.

If no fault-free component is available, the SFS should report its functional status to the reasoning unit to avoid the malfunctioning of the intelligent system as a whole. Due to the stringent timing requirements of the periodic and aperiodic events (Chapter-2), the overhead of the detection and clearance of these component faults should be estimated. This estimate will allow the developer to keep enough room in the temporal specification of the discrete requirements and the constituting computing components to ensure the correct temporal performance of the system.

The SFS is composed of multiple building blocks. The fault-tolerance of these building blocks can be achieved using different techniques. The redundancy overhead of these building blocks has different effects on the periodic and aperiodic events of the requirements model of the system. In general, different techniques to achieve fault-tolerance of different building blocks of SFS will be explained in this chapter with particular emphasis on the sensor module. The improvement of reliability and the required overhead for the incorporation of fault-tolerance in sensor module will be analyzed. Special emphasis will be given to the dependence of different levels of fusion on the reliability of the supporting sensors.

5.2 The Fault-Tolerance of the Building Blocks

The SFS is made of six major building blocks: sensors, analog processors (APs), analog to digital converters (ADCs), digital processors (DPs), memory modules (MMs), and I/O. These building blocks operate in the analog, hybrid and digital domains. Due to their uniqueness, the techniques and effects of detection and clearance of faults of these building blocks are not identical.

5.2.1 The Fault-Tolerance of the Sensors

From the perspective of fault-tolerance, a sensor can be modeled as an analog electrical signal source $g(t)$ as shown in Fig. 5.2. Some of the commercially available sensors have built-in preamplifiers and filters to sense very weak information in noisy environment. Therefore, there is a justification for modeling a sensor as signal source from a practical point of view. Here, it has been defined that physical information (e.g., pressure, vibration, temperature) will be called *physical signal* and its electrical equivalent will be called *signal*. The wave form of $g(t)$ depends upon the pattern of the sensed physical signal, $p(t)$, and the functioning state, $s(t)$, of the sensor as shown by the relation

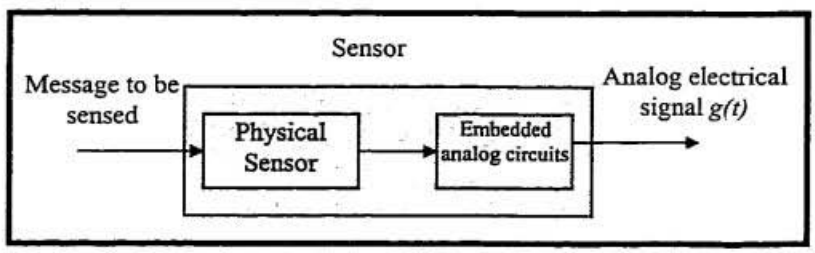


Figure 5.2: Model of a sensor as an analog signal source.

$$g(t) = Kp(t)s(t) \tag{5.2}$$

Here, the coefficient K, the conversion ratio, is a constant. From the perspective of fault-detection (i.e., presence of fault), $s(t)$ is a binary signal: erroneous value and error-free value.

5.2.1.1 Techniques of Fault-Tolerance of Sensors

To achieve fault-tolerance in the operation of the sensors, the faulty sensors should be detected and replaced by fault free ones. Through a simple switching mechanism a faulty sensor can be replaced by a properly functioning one. The main problem lies in the detection of a faulty sensor. The state of the functioning of the sensors can be detected by using majority voting and estimation techniques [55],[57],[30].

5.2.1.1.1 Majority Voting Technique for Sensor's Fault Detection

The majority voting technique in fault detection of sensors is based on the principle that if a set of sensors S={s_i: 2≤ ⋮) is used to sense the same physical parameter, by comparing the sensed signals from different sensors the state of the sensors can be determined. This phenomenon is explained in Fig. 5.3. Experimentally faulty sensor was detected using this voting concept as reported in section G.2. Different types of voting techniques are used in the comparator module to detect the state of the sensors. Available voting techniques work satisfactorily if the size of the set of the redundant sensors is more than two [57]. In case of two redundant sensors, the voting technique is able to detect only the presence of faults, but is unable to detect the faulty sensor as shown in Fig.5.4. This technique completely fails to detect the state of the

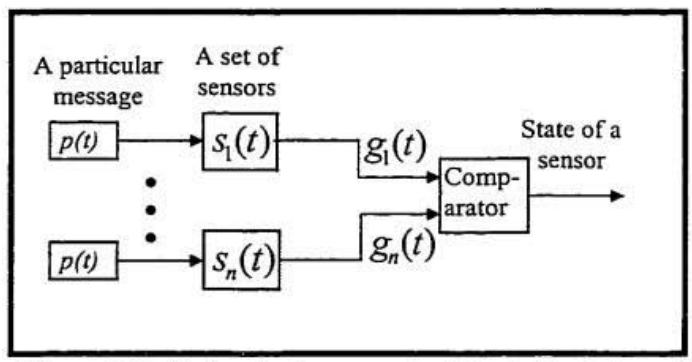


Figure 5.3: Redundant sensors to detect the states of sensors.

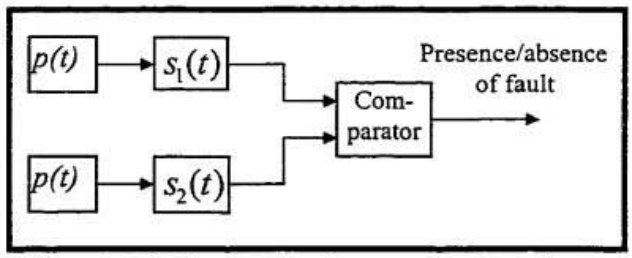


Figure 5.4: Two redundant sensors can detect only the presence of faults.

sensor if the size of the set is one. The state diagram using Markov's Model showing the possible state transitions for a triple modular redundant sensor system using voting as a fault detection technique is shown in Fig. E.4. The reliability profiles of fault-tolerant sensor systems having triple and 4-modular redundancy are shown in section E.3. From these profiles it is evident that a sensor module with redundant sensors has the potential to have higher reliability than that of single sensor alone.

The addition of the comparator module (usually in hardware) increases the complexity of the system. The failure of this module will result in the failure of the system. Thus the reliability of the system will suffer. To make this scheme effective, the redundant sensors should be in operation. This operational requirement will make the effective service time of the sensors shorter. Moreover, in a power constrained operation (e.g., unsupervised autonomous system in microgravity powered by battery) the extra power loss in the operation of redundant sensors will impede the application of this scheme. The cost related to deploy extra sensors is another limitation of this scheme. This scheme becomes ineffective and even can provide a completely wrong decision if more than half (or all) of the sensors fail simultaneously. This situation may occur due to a problem in power supply. The overheads in the form of hardware, energy, and space to incorporate fault tolerance in the sensor module of an example sensor fusion system are explained in Appendix E.2.1.

The saturation characteristic of the physical sensor and the supporting electronics is another constraint to the effectiveness of this approach. If the values of the physical signal go beyond the expected threshold value, due to the saturation effect of the sensor, the signal generated from the sensor will not represent the corresponding physical signal. In this operating condition, all the sensors will generate the same $g(t)$(dc value due to saturation effect). Therefore, this voting technique will fail to detect that the sensors have failed to convert the physical signal $p(t)$ to its equivalent electrical signal $g(t)$.

5.2.1.1.2 Estimation Technique for Sensor's Fault Detection

The message space *M*, the state of the sensor, has two messages m_1, fault-free state, and m_2, faulty state and the decision space *D* also has only two elements d_1, sensor is functioning properly, and d_2, sensor is malfunctioning. Therefore, this is a binary decision theory problem [66] and the decision is based on the measure of differences between the features of the *p(t)* and *g(t)*. If the probability distributions of the features of *p(t)* are known, the decision rule *d(z)* maps the observation space, features of *g(t)*, into the binary decision space in some optimal manner. Since there are only two decisions, this is equivalent to dividing Z into two decision regions Z_1 and Z_2 such that

$$d(z) = d_1 \text{ if } z \in Z_1 \tag{5.3}$$

$$d(z) = d_2 \text{ if } z \in Z_2 \tag{5.4}$$

The regions Z_1 and Z_2 must be disjoint (i.e., $Z_1 \cap Z_2 = \varnothing$) in order that each point in Z will yield a unique decision. The regions Z_1 and Z_2 must cover Z (i.e., $Z_1 \cup Z_2 = Z$) in order that each point in Z will have a decision associated with it.

If the adopted estimation technique satisfies these decision conditions, most of the limitations to detect sensor fault using different voting techniques will be overcome. The limitations of this scheme are the degree of accuracy of its ability to detect sensor faults and the required computational time to perform this detection.

The comparisons of the reliability profiles of a triple and four modular redundant fault tolerant sensor systems using voting and estimation techniques as a means of fault detection are given in section E.5. The ratios of the reliability profiles of fault tolerant systems using voting and estimation techniques shown in Fig. E.9 clearly indicate that estimation based technique has the potential to achieve higher reliability than voting.

5.2.1.2 The Effect of Sensor's Fault-Tolerance on the Performance of the System

The periodic events (Chapter 2) interact with the sensors. Therefore, the overhead related to the fault-tolerance of the sensors will affect the periodic performance of the system. The system waits in the readiness state to serve the next periodic event. To make the operational scenario comprehensible, the interval between two successive periodic events is divided into multiple segments as shown in Fig. 5.5. The relationship among these segments is given by the following equations

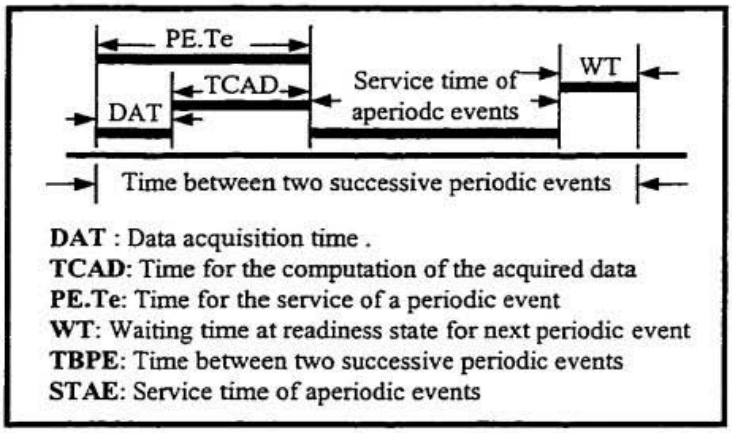


Figure 5.5: Distribution of time for different tasks between two successive periodic events.

$$TBPE = DAT + TCAD + STAE + WT \tag{5.5}$$

$$PE.Te = DAT + TCAD \tag{5.6}$$

The occurrence of a sensor's fault during *TCAD*, *STAE*, and *WT* does not interfere with the operation of the tasks executed during these periods. Therefore, the detection and clearance of sensor's fault occurred during these periods can be deferred to the beginning of the next periodic event. The value of *WT* is random, which varies from zero to a certain positive number depending upon the operating conditions of the system. Therefore, fault-clearance during this period may not be justified for safety critical systems. However, for systems with limited safety requirements, the developer can use this time for the clearance of faults. The conservative solution is to extend the *DAT* to accommodate time for the detection and clearance of faults of the corresponding sensor. Unfortunately, this

scheme wastes system resources. Another scheme is to pause the periodic event generation clock to extend the *DAT* on demand basis. This scheme dynamically changes the sensing period. If both the rate of occurrence of faults and the time required for detection and clearance of the faults are very low, this scheme is an acceptable approach. There is a need for optimization to select a particular approach.

If fault occurs during DAT, the fault should be cleared immediately. During this fault clearance period data will be lost and the DAT will be extended due to this interruption. Therefore, steps should be taken to address this problem to handle faults occurred during DAT.

The effects of fault-tolerance overhead on the system depend upon the adopted fault detection scheme (i.e., voting and estimation). Therefore, they should be treated separately.

5.2.1.2.1 The Effect of Voting Technique

If the voting algorithm is implemented in hardware logic in the comparator module, the fault detection scheme will run in parallel with the service of other events creating no temporal overhead on the execution of the user's requirements. Therefore, it is recommended to extend *DAT* to clear the faults occurred during *TCAD, STAE, and WT.* But if faults occur during *DAT*, valuable data will be lost during fault clearance time as shown in Fig. 5.6. The presence of this effect has been detected in an experimental setup to achieve fault-tolerant triple modular optical sensor as shown G.2. Either the data must be restored or

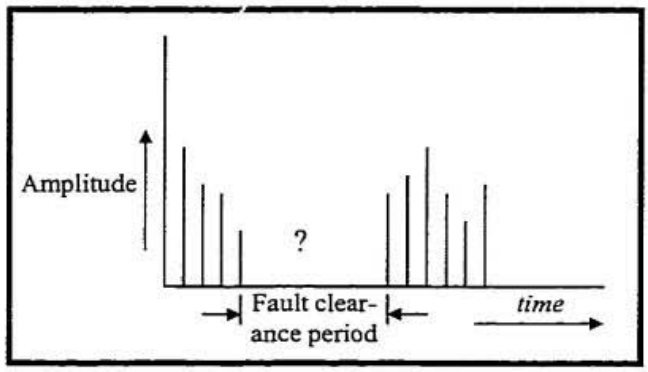


Figure 5.6: Loss of data during fault clearance period

the data acquisition process must be repeated. Both the restoration and the repetition will extend the *DAT*. The developer has the option either to keep enough room (required for restoration or repetition) to specify the *PE.Te* or to pause the clock until error free data acquisition is complete to accommodate this extension. The decision will be driven by the nature of the system. For stringent temporal specifications, it is recommended to specify *PE.Te* considering the time required for data restoration. On the other hand, for less time critical systems it may be feasible to pause the clock until error free acquisition is complete.

5.2.1.2.2 The Effect of Estimation Technique

This technique analyzes the sensor data to estimate the state of the sensor. Since the acquisition of data is a precondition of this technique, part of PE.Te is permanently assigned to do this estimation job. If the fault is detected, the faulty sensor must be replaced by a fault-free one and the data acquisition process must be repeated. It should be noted that the fault detection task is executed in sequential fashion. The estimation technique is unable to separate the faults occurred during DAT and TBPE-DAT. In this scheme there is no room for data restoration. The estimation algorithm should be executed every time to acquire data from the sensor. Therefore, this will create significant overhead in the temporal behavior of the system. To reduce this effect, the estimation algorithm should be simple and effective. A very complex algorithm may be very effective in fault detection, but may not be suitable for application due to temporal overhead. An optimization of the effectiveness and temporal overhead should be performed to deploy any estimation technique for fault detection.

This basic framework to detect the overhead to incorporate sensor fault tolerance in system design has been used to detect temporal overhead of an example sensor fusion system as explained in Section E.7.

5.2.2 The Fault-Tolerance of the Analog Processors (APs)

The job of the analog processors is to condition the signals generated from the sensors to make them easily interpretable. The propagation of physical signal through the sensor and analog processor (responsible for signal enhancement) can be explained by Fig. 5.7. In a sensor fusion system, the information about $p(t)$ is recovered from $f(t)$. Therefore, it is essential that the wave shapes of $f(t)$ and $p(t)$ must be very similar. If it is assumed that the sensor is fault free, the relationship among $p(t)$, $f(t)$ and the operating state of the AP, $ap(t)$, is defined as follows

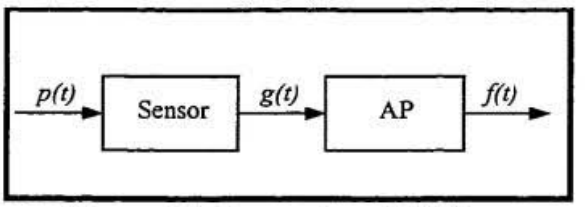


Figure 5.7: Generation of $f(t)$ from physical signal.

$$f(t)=K\,p(t)\,ap(t) \tag{5.7}$$

This Eq.(5.7) is similar to Eq.(5.2), which relates the physical signal, the operating state of the sensor and the output signal $g(t)$ from the sensor. Therefore, the fault detection techniques as described for the sensors can be applied for the APs. The analog processors like the sensors are used to serve the periodic events. Therefore, the effects of the fault-tolerance of the APs are similar to those of sensors.

5.2.3 The Fault-Tolerance of the Analog to Digital Converters (ADCs)

The ADC converts the analog signal, $f(t)$, conditioned by the AP into a digital signal, $f(nT)$. The information content of $f(nT)$ and $f(t)$ must be very close to make it possible to recover information from $f(nt)$ about $p(t)$. Therefore, the relationship between $f(t)$, $f(nT)$ and the operating state of the ADC can be explained in the same way as for the

relationship between sensor and the AP. Like sensors, Aps and ADCs are only used to serve periodic events. Therefore, their fault tolerance and effects on the operation of the system can be viewed in the same ways as those for sensors and APs.

5.2.4 Separation of faults of Sensors, APs and ADCs

If the voting technique is used to detect the faults in the sensors, the APs and the ADCs, the failure of these units generate unique events as shown by Fig. 5.8. If the estimation technique is used to detect the faults of sensors, APs, and ADCs, the problem is complicated as the estimation technique is based upon the difference of the features of $f(nT)$ and $p(t)$ as shown in Fig. 5.9.

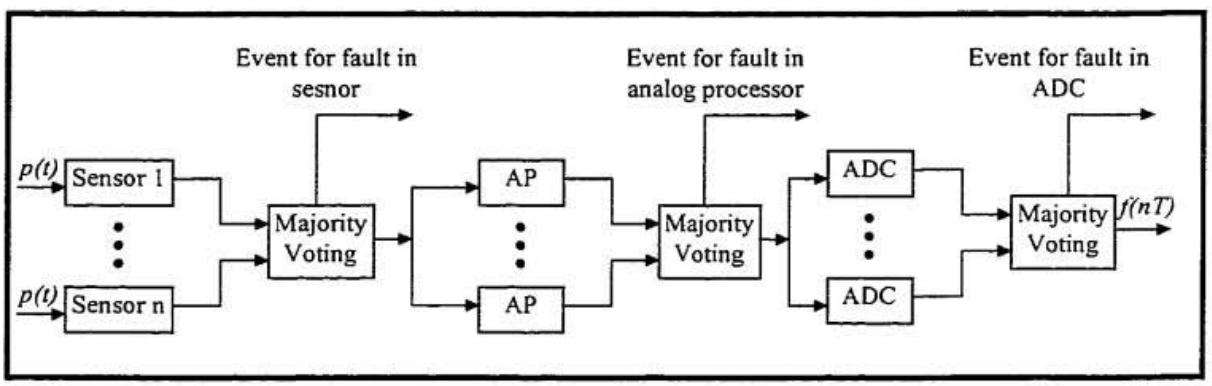


Figure 5.8: The generation of event for detection of faults in sensor, AP and ADC

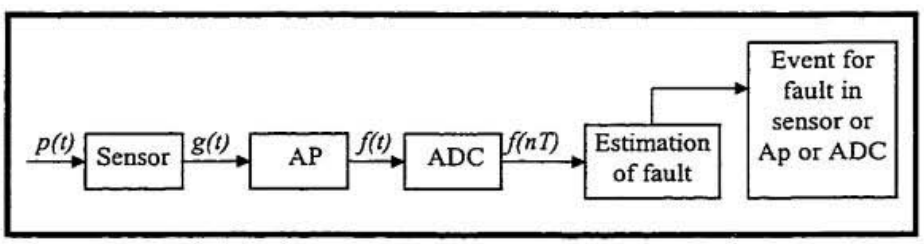


Figure 5.9: The generation of events for detection of fault in sensor, or AP, or ADC

Therefore, the faults of sensors, APs and ADCs are fused together as single fault. Under this circumstance, the sensors, APs and ADCs should be replaced one after another until the system is fault free. This will increase the fault detection overhead significantly. Therefore, to use estimation technique, this overhead should be considered.

5.2.5 The Fault-Tolerance of the Digital Processors (DPs)

Both voting and estimation techniques may be used to detect the faulty processor. The voting technique uses more than two processors connected in parallel through a comparator similar to the technique used for the sensor. After detection of a fault, the faulty processor should be replaced by a fault free one. The voting technique for detection of faulty processor suffers from similar limitations to those of the voting technique for the detection of faulty sensor. Therefore, these disadvantages should be considered before accepting technique.

The estimation techniques for the detection of faulty processors are different from those for sensors, APs and ADCs. The basic concept of estimation of the presence of faults in a processor is to test the functional reproducibility of the processor [57]. Allowing the processor to perform a known task can test this. The execution of this test job should use all of the internal components of the processor.

Different types of voting and estimation techniques are available in the public domain [55]. The objective of this work is not to develop a new technique or explain the publicly available ones, rather it is to study the effect of these techniques on the performance of the system.

5.2.5.1 The Effect of Digital Processor's Fault-Tolerance on the Performance of the System

The failure of the digital processor will affect both the aperiodic and periodic events unlike sensors, APs and ADCs. Therefore, special care should be taken to handle faults of the digital processor.

5.2.5.1.1 The Effect of Voting Technique

In this scheme, due to the occurrence of faults in the processor, any event (both aperiodic and periodic events) under execution will be interrupted. If a fault occurs during DAT, either data lost during the fault clearance time should be restored or the data acquisition process should be repeated until error free data are collected. If a fault occurs during the period other than DAT, only the problem of unpredictable delay of service of the events should be addressed. It has been assumed that due to the occurrence of faults in a processor all processing states can be restored from a parallely operated fault free processor. The delay of the service time of the events due to the temporal overhead of fault clearance can be addressed either keeping enough room (required for restoration or repetition) in the temporal specification of the discrete events or pausing the periodic event generation clock on demand basis.

5.2.5.1.2 The Effect of Estimation

Estimation is a delayed fault detection technique and in order to minimize the effect of delayed detection the test program should be run as frequently as possible. The malfunctioning of the system due to this delay should be as minimum as possible. This requirement will create extra computational overhead on the service of the events. If n is

the number of times the test program is run during the service of an event, the service time of the event will be extended by

$$\Delta Te = nT \tag{5.8}$$

Here, T is the execution time of the test program.

Therefore, during the requirements modeling this overhead should be considered to ensure temporal performance of the system. It should be noted that the effect of the delay of the detection of fault should be optimized with the execution overhead of the test program. The sequence of tasks to acquire fault-free data while estimation technique is used to detect faulty sensors is shown in Fig. E.24.

5.2.6 The Fault-Tolerance of the Memory Module (MD) and the digital I/O

The fault tolerance techniques for memory module and digital I/O are similar to those for the digital processor. The effect of faults of memory module and digital I/O are also similar to those for the processor. Therefore, these problems should be addressed as discussed in section 5.2.5.1.

5.3 The Measure of the Dependence of Different Levels of Fusion on the Reliability of Sensors

In a typical sensor fusion system, fusion occurs at three different levels: data, feature, and decision. To achieve the goal of this work to synthesize an engineering framework for the development of a reliable sensor fusion system, it is important to measure the dependence of a particular level of fusion on the availability of data from the supporting sensors. To materialize this objective, a fault tree [65] has been used to visualize the links of the failures of the sensors to the failures of the generation of the events at different levels of fusion. In this proposed discrete event frame work, fusion occurs through the service of these events. Therefore, a failure to generate these events will result in the failure of fusion. The fault tree of a typical event is shown in Fig. 5.10. In this proposed framework, the service of the event '**A**' will result in the fusion of data from sensors 1, 2, 3, and 4. The failure of the supply of data from the sensors 1 or 2, 3, and 4 will result in the failure of the generation of the event '**A**'. The reliability of the generation of the event 'A' (i.e., fusion of data by the service of event 'A') is a measure of the compound reliability of these sensors. The quantitative relation to measure this reliability profile is shown below.

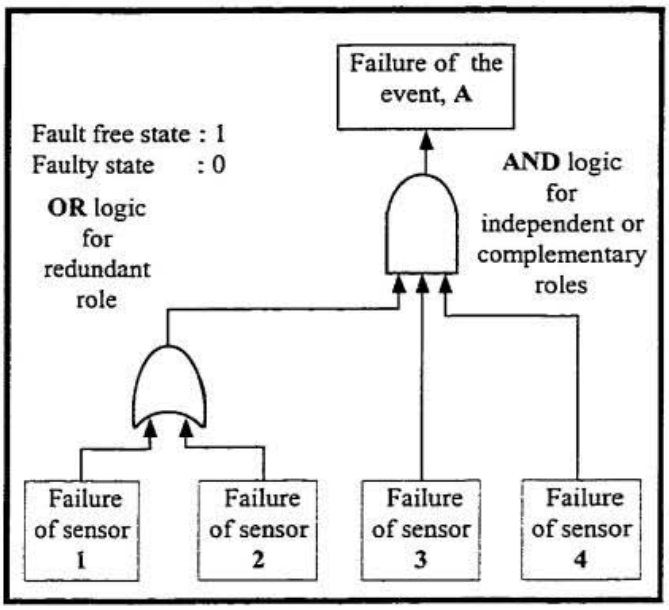


Figure 5.10: The fault tree of a typical event.

$$R(t) = [1 - \{1 - R_1(t)\}\{1 - R_2(t)\}]R_3(t)R_4(t) \tag{5.9}$$

Here, $R(t)$ is compound reliability; $R_1(t)$, $R_2(t)$, $R_3(t)$, and $R_4(t)$ are the reliabilities of sensors 1, 2, 3, and 4 respectively.

This development has been applied for the depiction of the fault trees and reliability profiles of different aperiodic events of the example sensor fusion system (Appendix A and B) at different levels of fusion. This SFS fuses data at three different levels (e.g., data level, feature level, decision level) by the service of eight aperiodic events (i.e., AE_1,...,AE_8).

5.3.1 The Fault Trees and Reliability Profiles of the Example Sensor Fusion System at Data Fusion Level

The data level fusion is performed by the service of the five aperiodic events: AE_1, AE_2, AE_3, AE_4, and AE_5. The fault trees of the generation of these events AE_1, AE_2, AE_4, and AE_5 are shown in Fig. E.10, Fig.E.16, Fig.E.18, and Fig.E.12 respectively. The reliability profiles of these events with different levels of redundancy are shown in Fig.E.11, Fig.E.17, Fig.E.19, and Fig.E.13 respectively. From these reliability profiles it is evident that the redundancy in the sensor modules decreases the probability of failures of these events.

5.3.2 The Fault Trees and Reliability Profiles of the Example Sensor Fusion System at Feature Fusion Level

The generations of the aperiodic events AE_6 and AE_7 cause the feature level fusion. The fault trees of AE_6 and AE_7 are shown in Fig.E.20 and Fig.E.22 respectively. The reliability profiles of these events shown in Fig.E.21 and Fig.E.23 make it clear that the fault tolerance in the sensor system has the potential to enhance the reliability of feature fusion.

5.3.3 The Fault Trees and Reliability Profiles of the Example Sensor Fusion System at Decision Fusion Level

The fault tree of the only event AE_8 responsible for decision fusion in this sensor fusion system is shown in Fig.E.14. The reliability profile of this event shown in Fig.E.15 shows that the redundant sensor system also increases the reliability of decision fusion.

5.4 Chapter Summary

The potential to enhance the reliability of sensor system by the use of fault tolerance has been evaluated. The overheads to incorporate this fault tolerance have been identified. Techniques have been proposed to change the DEVR model to accommodate this overhead. The fault-tree based approach is proposed to measure the reliabilities at different levels of fusion and this technique has been used successfully to evaluate the reliability profiles at different levels of fusion of an example sensor fusion system as shown in Appendix E. This novel technique to measure the reliability of different levels of fusion in terms of the reliability of the supporting sensors has the potential to improve the state-of-the-art of the engineering method of multisensory sensing systems. The measurement of this critical design parameter is based on sound quantitative reasoning and tested in the design of an example sensor fusion system. This knowledge component specifically contributes to the process of synthesization of scientific knowledge to engineer highly reliable sensor fusion system.

Chapter 6

The Detection of Sensor Faults Using Local Statistics

6.1 Introduction

High reliability is a precondition for the deployment of multi-sensori automated systems in safety and mission critical operations. The use of redundant sensors has the potential to enhance the reliability of the sensing sub-systems (Appendix A). The estimation based fault detection scheme has better potential to enhance the reliability than voting technique based fault detection scheme (Section E.5). Moreover, the voting technique fails to detect the presence of faults if all the redundant sensors are affected [57], for example in the case of transient faults due to the switching actions of the neighboring inductive loads (e.g., dc motors, relays) [67] or due to electrostatic discharge induced in a space environment. These transient faults may corrupt the data acquired from multi-sensori unsupervised autonomous systems (e.g., scientific experiments deployed in unmanned space environment [55] or in other special environments). If the presence of transients remains undetected, the corrupted data acquired from these unsupervised autonomous systems may lead to the misconceptions about the sensing environments. To remain within the scope of this thesis, the detection of transient faults will be investigated in this Chapter of this thesis. Other types of sensor faults are not addressed by this work (e.g., intermittent fault).

In a single processor based multi-sensori system, data are acquired from each sensor for a short period of time in a particular sequence. The sequence of sensing is defined by the phases and the periods of the sensors defined by the DEVR model of the SFS (section 2.2). The data acquired from each sensing session must be interpreted to detect the presence of faults. The main aim of this research is to devise a robust mechanism to detect the presence of transient faults in the

acquired data. As reported in Section 5.2.1.2, the data interpretation time required fault detection must be as small as possible to keep temporal overhead at minimal level.

For a particular process, the measurable parameters have specific domains of amplitudes and variances. The prior knowledge of these domains may facilitate the detection of faults in the sensors deployed to measure these process parameters [68]. The detection of sensor faults by examining the pattern of deviations of engine signals from their nominal unfailed values has been reported [69], and a Kalman filter-based dedicated observer has been used to detect sensor faults [70]. The requirement for a dedicated processor for each sensor is a limitation for its use in single processor-based multi-sensori systems. Sensor faults have been detected by using computationally complex and model-based estimation technique [71]. The sensor signal amplitude has been used to detect the failure of faults in automotive engines [72]. The detection of transient fault has not been addressed [72]. The different algorithms based on analytical and knowledge based redundancy for fault diagnosis reported in a survey [73] are computationally complex. The development of another computationally complex algorithm has been reported to detect sensor faults [74]. The requirement for a dual-redundant system is the limitation of a reported technique to detect sensor faults [75]. The limited scientific basis (subjective due to the dependency on training set) for measuring the pattern recognition performance makes the neural network based fault detection approach [76] inappropriate for the present problem. A reported technique for the detection, isolation and identification of sensor faults in nuclear power plant does not address the problem of detection of transient faults [77]. The use of statistical characteristics to detect faults in earth sensors has not covered the detection of transient fault in sensor data stream [78].

The available techniques uncovered in this literature search have been developed mainly to detect the permanent faults of sensors. Despite the mathematical sophistication of these methods, it is fair to say that most of these techniques are computationally complex. Moreover, none of these techniques has addressed the problem of detection of transients present in sensor data in multi-sensori systems in a comprehensive manner. The requirement of a dedicated processor to detect fault in each sensor is another limitation of some of these methods. Therefore, it is fair to

conclude that these available techniques for the detection and isolation of faulty sensors are not computationally simple and adaptive. The objective of this part of this thesis is to develop a simple and adaptive technique for the detection of transients in sensor data stream. It should be noted that due to the potential of degradation of sensor signal the use of a low pass filter to eliminate the transient has not been pursued in this thesis.

6.2 The Detection of Transient Faults Using Local Statistics of Sensor Data

A typical sensor used to measure physical parameter produces an equivalent electrical signal corrupted with normally distributed random noise as given by

$$g(nT) = p(nT) + \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{(nT-\mu)^2}{2\sigma^2}} \tag{6.1}$$

Here, $g(nT)$ and $p(nT)$ are discrete sensor and physical signals respectively, and T is the sampling period.

The local variance and mean of a segment of this sensor signal are measured by the following relations:

$$g^2(K) = \frac{\sum_{i=0}^{N-1} (\overline{g(k)} - g((KN+i)T)}{N-1} \tag{6.2}$$

$$\overline{g(K)} = \frac{\sum_{i=0}^{N-1} g((KN+i)T)}{N} \tag{6.3}$$

Here, $g^2(K)$ and $\overline{g(K)}$ are the local variance and mean of the K segment.

The variance is the measure of the scatter of the corresponding sensor data (e.g., pressure, temperature) from the mean value [78]. The upper and lower limits of the variance specify the acceptable domain of randomness of the corresponding sensor data. The presence of a transient fault is characterized by the presence of variances outside of this domain. The use of this

parameter is proposed to detect the transient fault [79]. The transient is modeled here as damped sinusoid as explained in Section F.3 [67]. To verify the effectiveness of this approach, the local statistics of four test signals at different fault conditions are calculated by simulation and documented in Appendix F.

6.3 The Statistical Characteristics of the Test Signals

The waveforms of the four test physical signals and the corresponding sensor signals are shown in section F.2. Each test signal has been divided into 50 segments. The local statistics of each segment is calculated by placing a window of the same dimension at the beginning of each segment. The local means of these sensor signals resemble the waveforms of the corresponding physical signals. The upper bounds of local variances of the first three sensor signals are very low. Due to high frequency components of the fourth sensor signal the upper bound of the local variance of this signal is much higher than those of the first three sensor signals. The upper and lower bounds of the local means and variances of these four test sensor signals are shown in Table 6.1 and Table 6.2 respectively. The bandwidths of these test signals are shown in Table 6.3.

Table 6.1: The local means of test signals.

Test signals	Upper bound	Lower bound
g_1(nT)	7.01	1.82
g_2(nT)	8.20	2.91
g_3(nT)	7.01	1.82
g_4(nT)	7.60	3.72

Table 6.2: The local variances of test signals.

Test signals	Upper bounds	Lower bounds
g_1(nT)	0.1298	0.0057
g_2(nT)	0.1169	0.0065
g_3(nT)	0.1298	0.0057
g_4(nT)	1.006	0.0092

Table 6.3: Bandwidth of the test signals.

Signals	Bandwidth in Hz
g_1(nT)	20 Hz
g_2(nT)	40 Hz
g_3(nT)	60 Hz
g_4(nT)	180 Hz

6.4 The Analysis of the Signature of the Transient Faults on the Test Signals

A transient fault in the form of a damped sinusoid [67] of 5ms duration has been superimposed on these four test signals at 70ms from the origin as shown in section F.3.1, F.3.2, F.3.3, and F.3.4. It appears that the local means of these signals do not indicate the presence of these transients. The signature of these faults as sharp rise of local variances at almost 70ms from the origin reveals that the thresholding of local variances can be used to detect the presence of transients. The ratios of the peaks of these local variances to those of the corresponding fault free sensor signals are shown in Table 6.4. The locations of these variance peaks are also shown in this table. It appears that the ratios of these variance peaks are function of the highest frequency components present in these test signals as shown in Fig. 6.1.

Table 6.4: The statistics related to the signature of the transient on test sensor signals .

Test signals	Ratio of peaks	Location from origin
g_1(nT)	42	70ms
g_2(nT)	16	70ms
g_3(nT)	13.79	70ms
g_4(nT)	2.10	70ms

From this simulated test result, it is found that there is a good potential to detect and locate the presence of transients on sensor signals by comparing the peaks of the local variances at faulty condition with those at fault free condition. It should be noted that the achievable locational accuracy is extremely high and is not a function of the bandwidth of the sensor signals. It is found that the delectability of transients by this method diminishes with the increase of the bandwidth of the sensor signal.

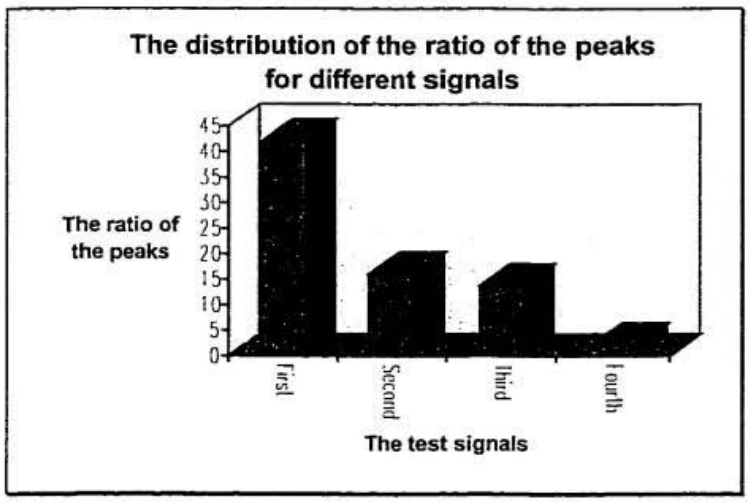


Figure 6.1: The distribution of the ratio of the peaks of local variances of the test signals at transient fault.

6.5 The Effect of the Transient Faults at Different Locations on the Local Statistics of the Sensor Signals

The occurrences of transients at 98 different locations on these test signals are studied. These faults have been simulated at integral multiples of 2ms from the origin. The profiles of the local statistics of the test signals with transient faults at different locations are shown in section F.4.1, F.4.2, F.4.3, and F.4.4. From this simulation results it is clear that the local means are not affected by the variations in faults locations. But the ratios of the maximum peaks of the variances vary noticeably with the variation of the location of occurrence of transient faults. The pertinent salient features of these variations are summarized in Table 6.5.

Table 6.5: The variation of the maximum peaks of the sensor signals with the occurrence of transient faults at different locations.

Sensor signals	Upper bound of the maximum peak	Lower bound of the maximum peak	Difference of variations	% of variation
g_1(nT)	43	34	9	32 %
g_2(nT)	20	14	7	50 %
g_3(nT)	18	13	5	38%
g_4(nT)	4	2	2	50%

Due to this wide variation of the peaks with the location of the transient faults, it is recommended to consider the worse case scenario to set the threshold. This simulation reveals that the ratios of these variations are not directly related to the bandwidth of the signals. But the differences in these variations are found related to the bandwidths of these test signals. From the profiles of variances of the occurrences of transients at two different locations from the origins (e.g., 0ms and 18 ms) it is understood that the locational accuracy is not a function of the position of the faults.

6.6 The Effect of Window Size on Local Statistics at Transient Fault on Test Signals

The local statistics of the sensor signals at different window sizes varying from .2 ms to 12 ms have been calculated. The simulation results of this study have been reported in sections F.5.1, F.5.2, F.5.3, and F.5.4. With a decrease in window size, the computational complexity increases linearly. From this simulation study it is found that with the decrease of window size the transient detectability increases due to the generation of higher value of variance peaks. At large window size the transient detectability decreases significantly. This finding can be clarified further by observing the effect of window size on the local variances of the fourth signal. At a .4ms window size, the maximum peak has a much higher value at the location of the transient than at other places. With the increase of the window size, values of the peaks at other places of the signal become comparable to those at the location of the transient resulting in a false detection. The effect of window size on the detectability of transients using this proposed local statistics based approach is shown in Fig.6.2. From this graph it appears that the probability of false detection of a transient increases with an increase in window size. It should also be noted that the peak size decreases with the decrease in window size beyond a certain value (e.g., the .6 ms for this test simulation). The maximum peak values for these four test signals have been detected at the .6 ms window size. Therefore, the dependence of the value of the peak on the window size must be taken into consideration in order to select the threshold for a particular sensor signal.

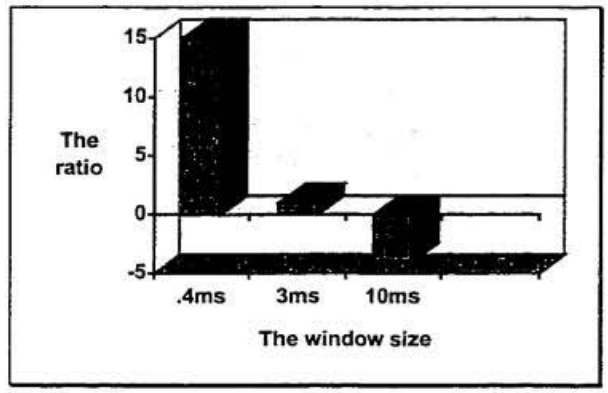


Figure 6.2: The variations of the ratios of the peaks during transient with the peaks during the rest of the signal at different window sizes for the fourth test signal.

6.7 The Effect of Window Locations Relative to the Position of the Transient

The window location has been varied in .05ms steps relative to the starting position of the transient. The simulation study reported in section F.6.1, F.6.2, F.6.3, and F.6.4 reveals that the value of the peak is a function of the location of the window. For these four test signals, it has been found consistently that the maximum peak size occurs when the window is located at the start of the transient. The variations of peak sizes with the change of the location of the window are shown in Fig. 6.3. Therefore, this characteristic must be taken into consideration in order to detect the transient using a local variance method.

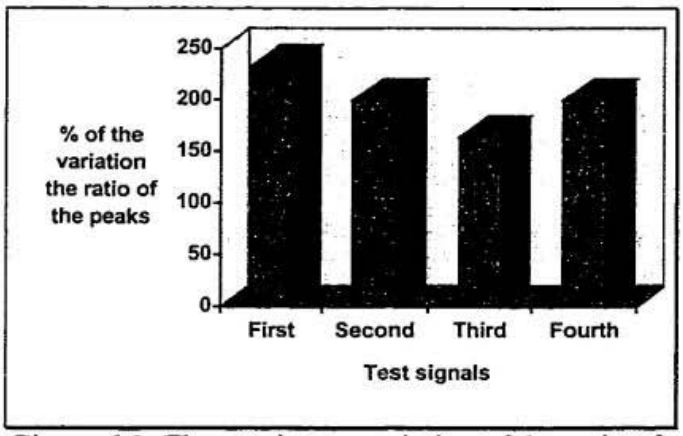


Figure 6.3: The maximum variation of the ratio of the peaks with the variation of the window locations relative to the transient.

6.8 The Effect of Different Frequencies of Transient Faults on the Local Statistics

The frequency of the transient has been varied from 500 Hz to 10 kHz in 500 Hz increments. The simulation results are reported in sections F.7.1, F.7.2, F.7.3, and F.7.4. From this study it is found that the detectability of transient increases with the increase of the frequency. The effect of the change of frequency of transient is more visible in low frequency sensor signal than in higher frequency sensor signal.

6.9 The Effect of Noise Power on Detectability of Transient Faults

It has been noticed that the detectability of transient faults using local statistics (e.g., variance) diminishes with an increase in noise power. The ratio of the maximum value of local variance of a sinusoidal signal (first test signal as shown in section F.2.1) to that of the same signal corrupted with transient noise is shown in Fig. 6.4.

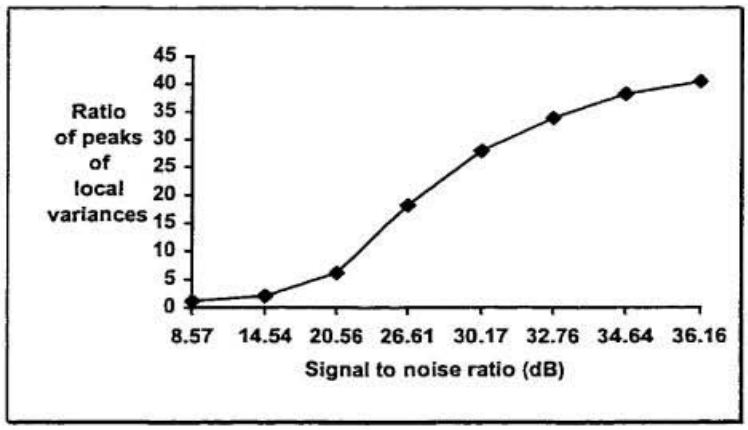


Figure 6.4: Variation of ratio of local variances with the signal to noise ratio(SNR).

6.10 The Detection of Permanent Faults

Under permanent faults, the possibility that the output may be stuck at the lowest value or stuck at maximum saturation level (intermediate values have not been addressed) has been addressed here. Due to these extreme low and high values of the sensor signal, the local mean will go beyond the normal operating domain. This deviation of the local mean at faulty condition than that at normal operating condition can be used to detect the faulty sensor. It has been shown through simulation in Appendix F that the local statistics mostly remain unaffected due to the occurrence of transient fault. Therefore, it would be possible to separate permanent faults from transient faults.

6.11 Chapter Summary

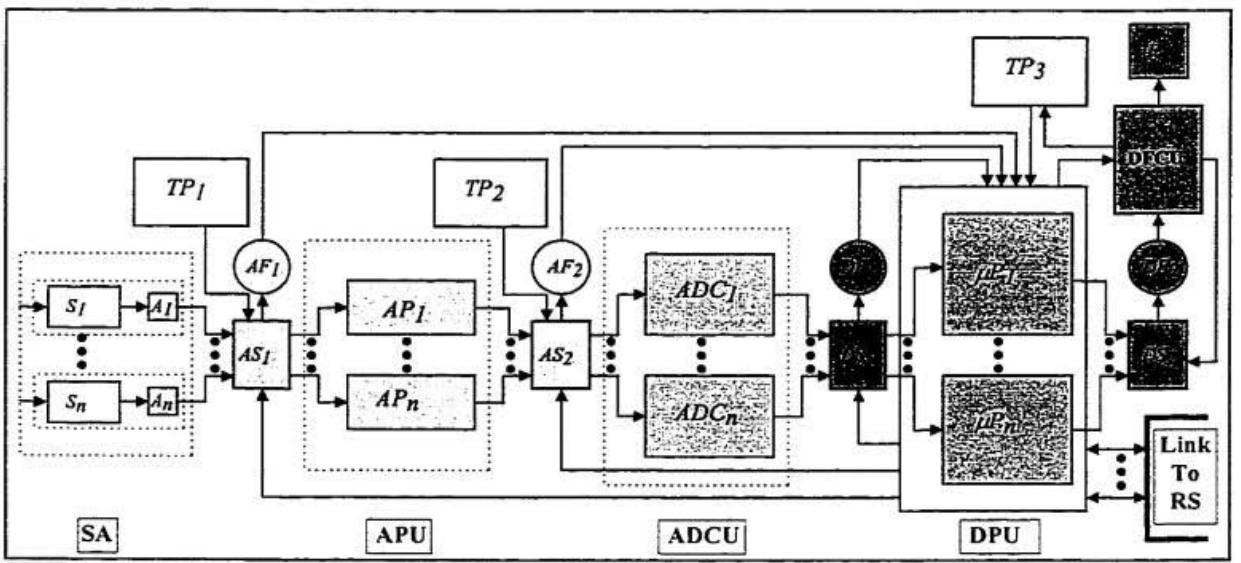
In this thesis it has been reported that there is a potential to detect and locate transient sensor faults sensor signals based on the local variance of the sensor signals. The detectability based on this principle is a function of the location of the transient, the window size, the location of the window relative to the starting of transient and the frequency of the transient. Through simulation study, the quantitative information of these relationships has been developed in this work. It has been shown that particular type of permanent fault can be detected with the information of local mean which is virtually unaffected by the transient fault. It should be noted that the estimation based sensor faults detection technique is based on finding odd features in the sensor signals. There is always a potential that the estimation based technique can falsely identify odd features.

Chapter 7

Restoration of Lost Sensor's Data During Fault-clearance Intervals

7.1 Introduction

The application of voting technique to detect faults in the fault tolerant sensor fusion system as shown in Fig. 7.1[80] will result in loss of data during the fault-clearance time(sec. 5.2.1.2.1). In a simplified form, Fig.7.2[56] shows the fault-tolerant sensing scheme with hardware redundancy. In this figure, $p(t)$ is the physical signal and $f(nT)$ is its equivalent digital electrical signal. The detection and replacement of faulty



SA : Physical Sensor arrays
$PS_1,\ldots, PS_n$ are physical sensors.
$A_1,\ldots, A_2$ Embeded amplifiers.
AS_1 : First analog switch
AF_1 : First fault detection unit in analog state.
TP_1, TP_2, and TP_3 are test pattern generators. .

APU: Analog signal processing unit
$AP_1,\ldots,AP_n$ are analog signal processors.
AS_1 : First analog switch
AF_1 : First fault detection unit in analog state

ADCU: Analog to digital conversion unit
$ADC_1,\ldots,ADC_n$ are analog to digital converters.
DS_1 : First digital switch
DF_1 :First fault detection unit in digital state.

DP: Digital data processing unit
$\mu P_1,\ldots,\mu P_n$ are microprocessors
DS_2: Second digital switch
DF_2: Second fault detection unit in digital state.
DFCU: Fault clearance unit of DPU
RS : Reasoning sub-system.

F: Flag to represent the status of the SFU.

Figure 7.1: An architecture of fault tolerant sensor fusion system [80].

components with fault-free components require certain amount of time known as the fault clearance interval. During this time, on-line sensor data will be lost. A simple solution to this problem is to repeat the data acquisition cycle. If the signal is highly transient, the repetition of the acquisition cycle will lose significant amount of information. Moreover, the unpredictable repetition will create significant overhead to satisfy the stringent timing requirements of the system. This unpredictable behaviour may result in malfunctioning of the system. Therefore, a scheme should be developed to restore the lost data.

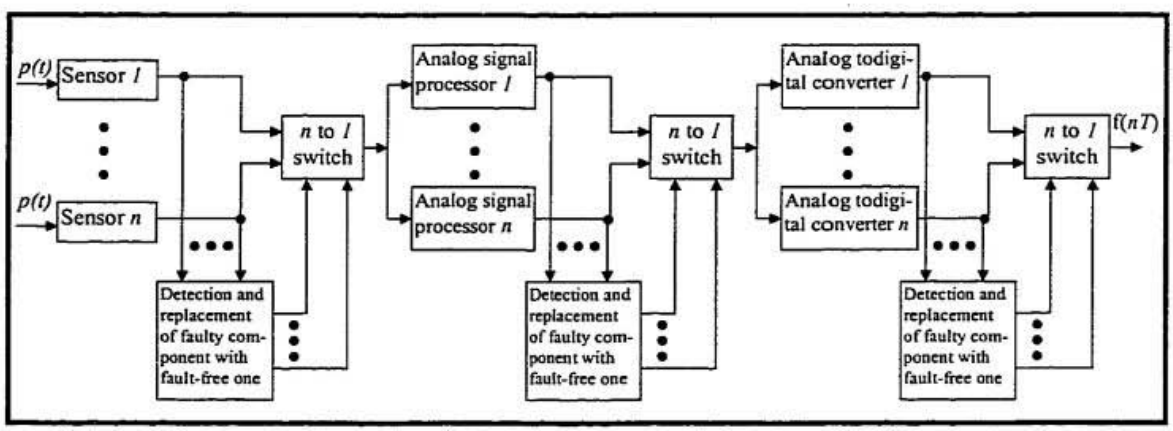


Figure 7.2: A general scheme of fault-tolerant sensing using hardware redundancy.

The restoration of lost samples in digital signals in the area of communication and digital storage is based on the estimation of the unknown samples from the information of the neighboring samples [81]. The methods documented in public domain literature deal with the restoration of samples of band-limited and low-pass signals [82]-[85]. If the signal is highly transient, the performance of these schemes suffers significantly. The estimation schemes having the ability to recover transient signals with reasonable performance are computationally complex [84]. Usually these schemes are iterative in nature. Initial estimates for unknown samples are chosen, the signal is restricted to its assumed frequency band, and the signal values at the positions of the unknown samples are used as new estimates. This procedure is repeated until satisfactory results are obtained [86]. The recovery of lost samples as solution of unknown samples from a linear system of equations has also been used [82]. Some of these schemes are non-adaptive due to the

requirements of signal spectrum, or equivalently the autocorrelation function has to be known in advanced [86],[87]. It appears that these available digital samples restoration schemes are not very effective to recover sensor data lost during the fault-clearance interval. To overcome these limitations, this thesis has proposed a new scheme based on parallel sensing to restore data lost during the fault-clearance interval in hardware redundancy based fault-tolerant sensing. This scheme is computationally very simple, non-iterative and is not limited to particular class of signals. It does not require any information about the nature of the signals and is virtually independent of the information for the neighboring samples. This proposed scheme is capable of restoring lost sensor data during fault-clearance interval.

7.2 A Unified Approach to Restore Lost Samples During Fault-Clearance Intervals

In a data acquisition session, N samples are acquired from a sensor. Let $s[n]$: $n=1,2,...,N$ be the segment of samples acquired in a particular session and **s**, the vector in which the segment of data is arranged. Let us assume that two segments of data $s_1[n]$ and $s_2[n]$ are collected in parallel from two separate sensors and analog channels sensing the same physical signal as shown in Fig. 7.3. Therefore, under fault-free operation, these two segments are identical. When a fault occurs, these segments of data contains F_1 and F_2 number of fault-clearance intervals. The positions of the samples during these intervals are at $t_{ij}(k)$, $k=1,2,...,m_{ij}$ and $j=0,1,...,F_i$. Here, i is the channel number, j is the number of fault-clearance interval and k

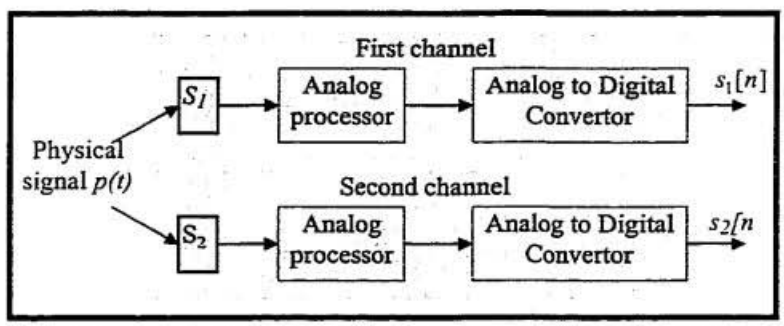


Figure 7.3 : The acquisition of data related to the same physical signal using two parallel channels

is the number of lost samples in the corresponding fault-clearance interval. It is assumed that these segments have o_i regions having undefined samples of length U at locations starting at the positions $l_{ir}[u]$, $r=0,1,...,oi$ *and* $u=0,1,..,U$. It has also been assumed that the fault-clearance intervals of these two channels do not overlap (i.e., at least one of these two channels is functioning properly during the entire data acquisition period). The samples in the *ith* channel during the fault-clearance intervals can be processed by the following equations

$$s_i[n]=0, \text{ if } n \equiv t_{ij}(k) \text{ for all the permissible values of } i,n,j, \text{ and } k. \quad (7.1)$$

$$s_j[n]=2*s_j[n], \text{ if } n \equiv t_{ij}(k) \text{ for all the permissible values of } i,n,j, \text{ and } k. \quad (7.2)$$

Most of the lost samples can be recovered by the following relation

$$s'[n] = \frac{s_1[n] + s_2[n]}{2} \quad (7.3)$$

Due to the switching effect, data at locations $l_{ir}[u]$ in this recovered segment will be undefined. Due to the very short duration of switching time the value of u will be very small. Therefore, these undefined data can be recovered by simple linear interpolation without incorporating significant error to obtain the estimated signal. This data recovery scheme can be explained by a simulated test signal as shown in Fig.7.4. This concept has been successfully used to recover data lost during fault clearance intervals in a laboratory experiments as shown in section G.4.

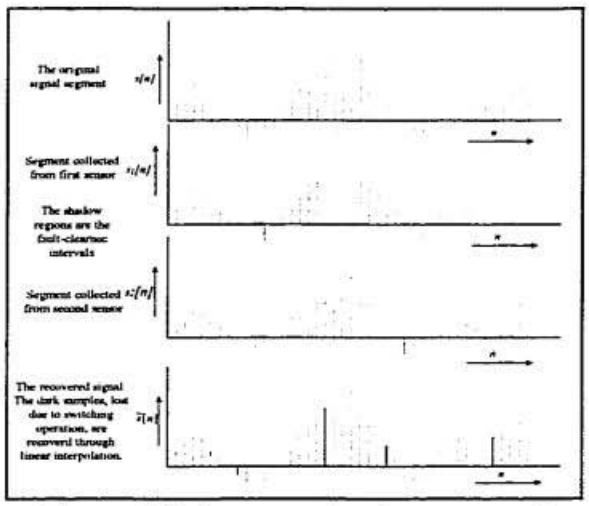


Figure 7.4: An example of recovery of samples lost during fault-clearance intervals.

7.3 Restoration in Fault-Tolerance with Dual Modular Redundancy

In dual modular redundancy, the voting technique cannot detect the faulty block. The presence of faults in any one of the constituting blocks (e.g., sensors, analog processors, and analog to digital converters) detected by the voting technique must initiate a process to detect and change the faulty block. The time at different steps of the fault-clearance process should be recorded to detect corrupted sample positions, so that sample values of those positions can be restored using the restoration technique as explained in the previous section (sec. 7.2).The fault-clearance process is explained in Fig. 7.5. Let us consider that the probabilities of failure of the same type of blocks in both the two channels are the same. It is assumed that only one block fails at a time and no other block fails during the subsequent fault-clearance interval. Since the fault-clearance time is extremely small in comparison to the total operating period of the system, this assumption is reasonable. It is also assumed that fault-free blocks are available for the fault-clearance operation. It should be noted that if the replacement of a probable faulty block in the first channel does not clear the fault, it is likely that the fault has occurred in the second channel. Then the fault-free block must be switched back to its previous position. This switching in the fault-free channel (in this case channel 1) will make a few samples during these transitions undefined. The transitions should be made as small as possible so that these undefined samples can be recovered using simple interpolation techniques (e.g., linear interpolation) without introducing significant error in the reconstructed signal.

The restoration of corrupted samples due to fault-clearance is shown in Fig. 7.6. It has been assumed that the interrupt generated for fault-clearance will carry the time of occurrence of fault. It should be noted that if f_S is the sampling frequency of the signal,

the delay (i.e., the value of m in delay block) and the fault-clearance interval should maintain the following relation

$$m \times \frac{1}{f_s} \geq \textit{fault-clearance interval.} \tag{7.3}$$

This means that both the two sequences must be stored before reconstruction.

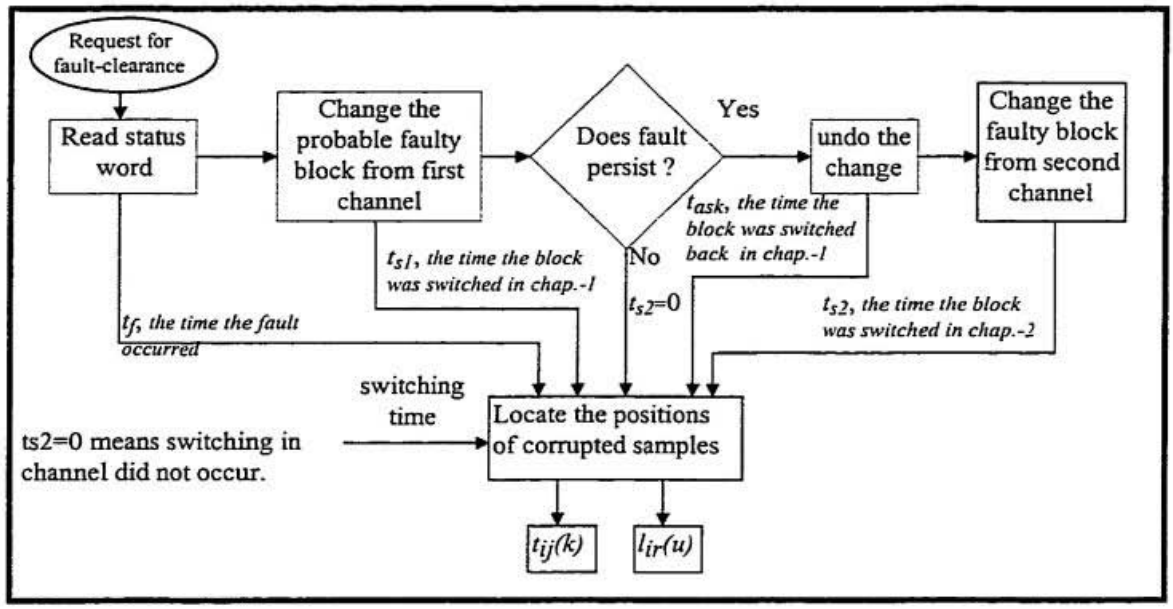


Figure 7.5: The flow diagram of the fault-clearance process.

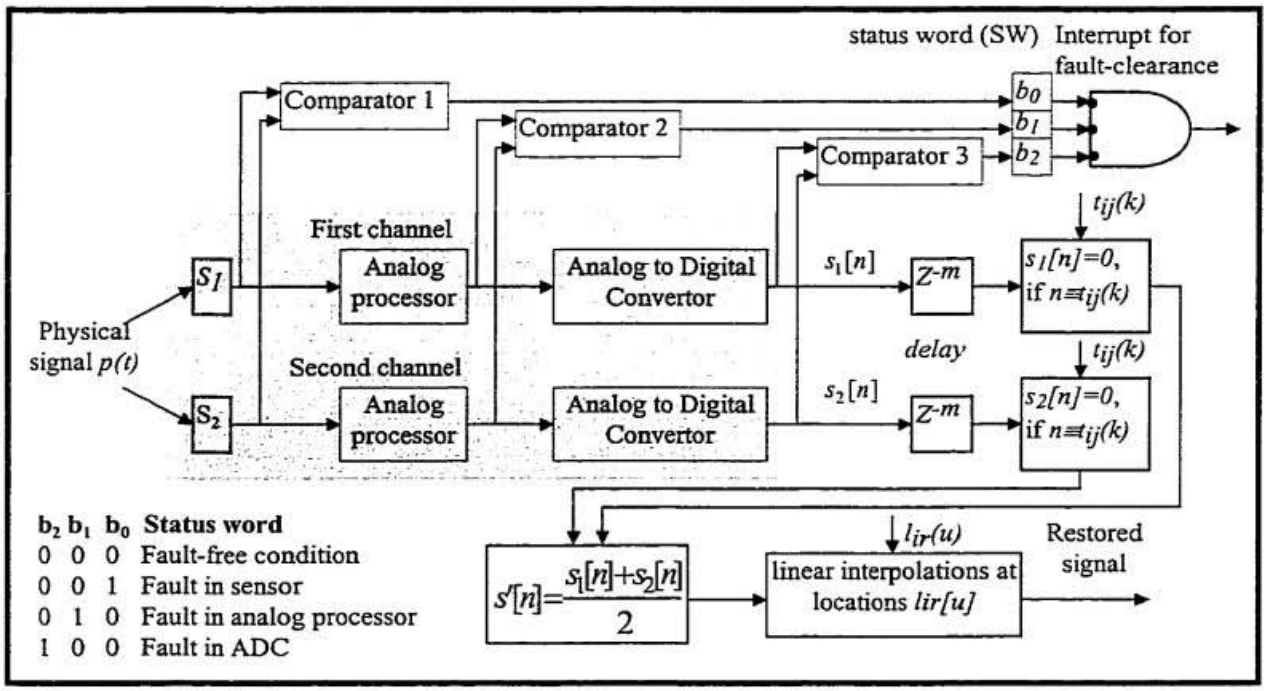


Figure 7.6: Restoration of data during fault-clearance in dual redundant fault-tolerant sensing.

7.4 Restoration in Fault-Tolerance with Triple Modular Redundancy

In triple modular redundancy, the voting algorithm to detect and replace the faulty block can be implemented either in hardware or in software. Usually the fault-clearance time is longer using software implementation than using hardware implementation. The fault-clearance time in hardware implementation can be made within the range of a few (e.g., one or two) sample times. The samples lost in this low fault-clearance time can be restored through simple interpolation techniques, such as linear interpolation. The restoration techniques in both the two approaches of implementation of voting algorithms are explained in the subsequent sections.

7.4.1 Restoration Using Hardware Implementation of Voting Algorithm with Triple Modular Redundancy

The basic concept of restoration of samples using hardware implementation of voting algorithm with triple modular redundancy in fault-tolerant sensing is shown in Fig. 7.7. For this reason it is assumed that the voting module senses three inputs and produces two outputs as shown in Fig. 7.8. The input signals are derived from the corresponding block as shown in Fig. 7.9. These signals are analog for the sensors and analog processors and digital for the analog to digital converters. The relationship between the two output signals and the selection of the corresponding block is shown in Fig. 7.8.

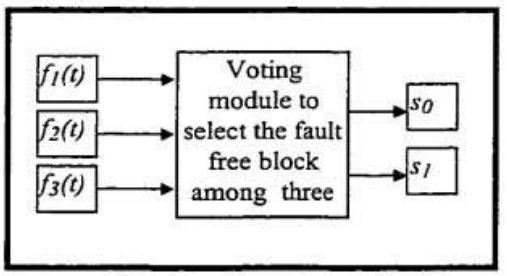


Figure 7.7: A simplified representation of voting module.

s_1	s_0	selected module
0	0	1
0	1	2
1	0	3
1	1	x

Figure 7.8: The selection of module with the output from the voting module.

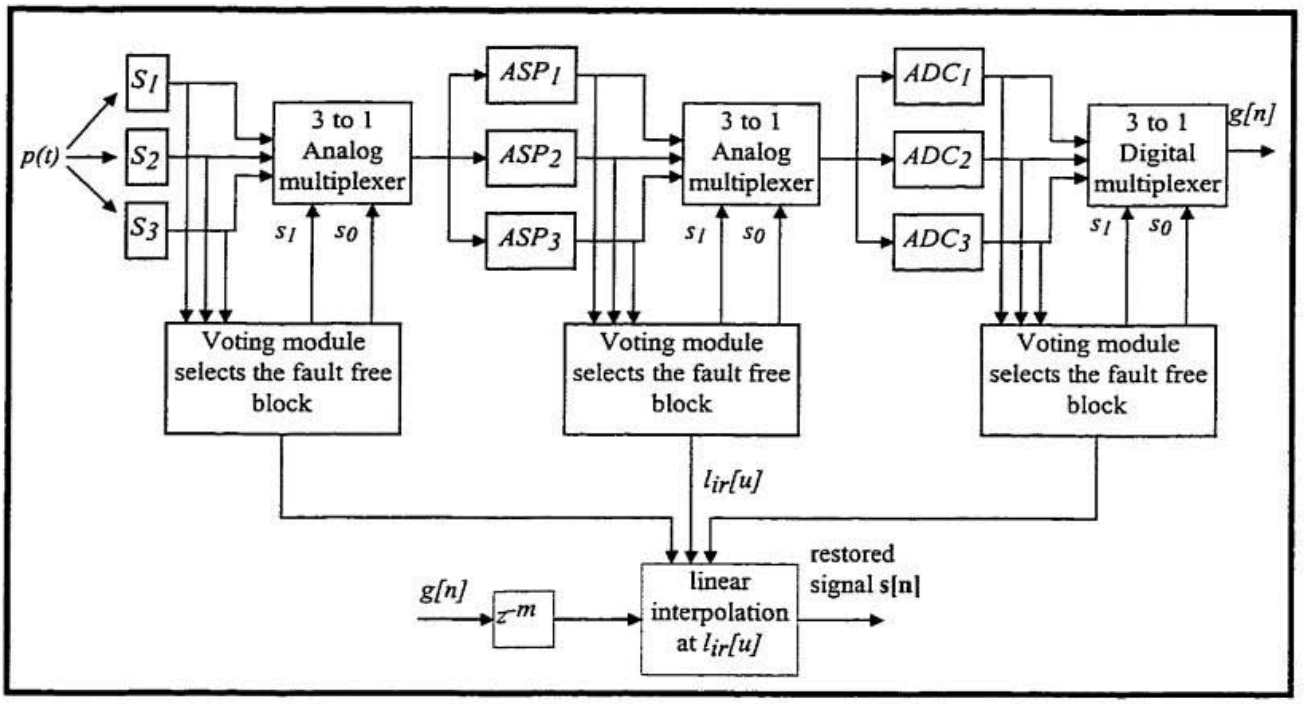


Figure 7.9: The restoration of lost samples in hardware implementation of the voting algorithm.

The input signals of the voting module are to detect the presence of fault. Analog comparators are used for the detection of faults of sensors and analog processors. For the detection of faults of ADCs, digital comparators are used. The comparator module and the relationship of the comparator module with the rest of the voting module is shown in Fig. 7.10. The fault-free block is selected from among the three modules based on the output of the comparator module using majority agreement of the inputs.

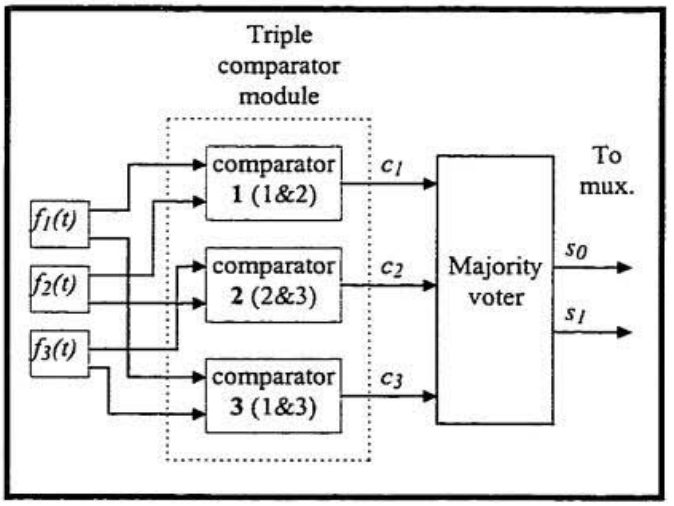


Figure 7.10: The comparator and majority voting modules in the voting module.

The output of a comparator is zero if its inputs are equal (within certain range) and is one if the inputs are not equal. The inputs and outputs of the majority voter are binary. Therefore, the voter can be designed as a digital combinational circuit. The truth table for the majority voter is shown in Table 7.1. If only one module fails, the voting algorithm is capable of generating the corresponding output signal to the multiplexer to switch to the next available fault-free module. If more than two modules fail, the output (1

Table 7.1: Generation of outputs from the voting module in response to the inputs from the comparators.

Input to the voter from the comparator			Output of the voter to the mux.		Fault-free modules
C_3 (1&2)	C_2 (2&3)	C_1 (1&3)	S_1	S_0	
0	0	0	0	0	1, 2, & 3
0	0	1	x	x	unrealistic
0	1	0	x	x	unrealistic
0	1	1	0	0	1&2
1	0	0	x	x	unrealistic
1	0	1	0	1	2&3
1	1	0	1	0	1&3
1	1	1	1	1	unknown

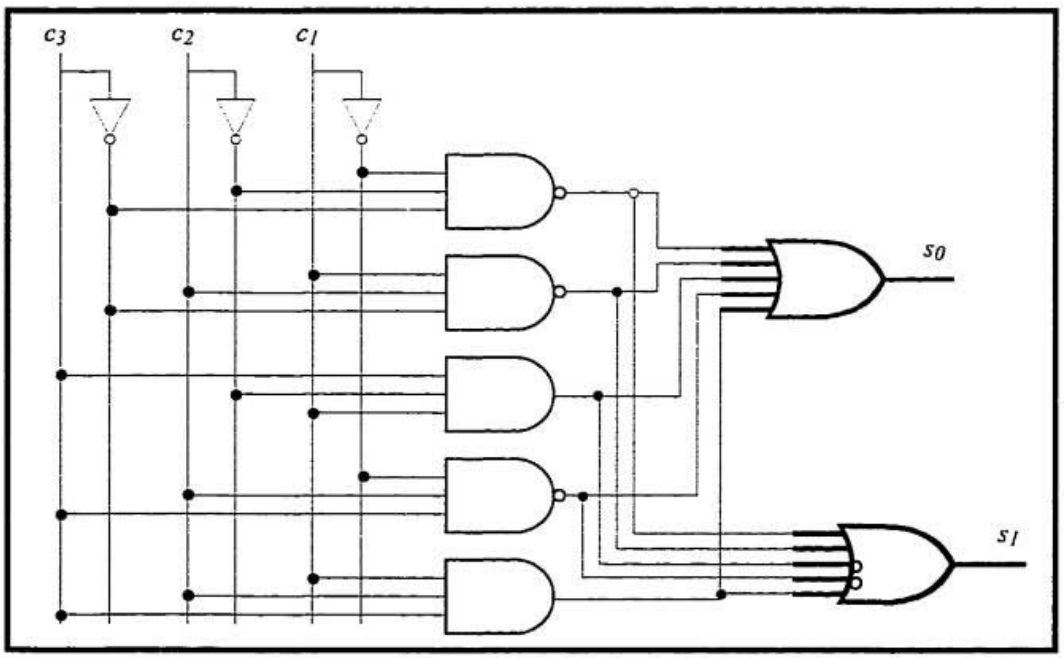


Figure 7.11: Hardware realization of the voting logic.

, 1) is an indication that more than two modules have failed. The digital logic circuit to implement this algorithm is shown in Fig. 7.11.

In this scheme, the multiplexers select one among the three inputs as output. This selected one can be called the main module and the remaining two can be called support modules. If the fault occurs in the main module, the hardware realization of the voting logic generates a command to the multiplexer to select a fault-free support module. After switching to the fault-free module, the voting unit interrupts the processor to replace the faulty module with spare fault-free module. The outputs of the comparator module (i.e., c_1,c_2, and c_3) are used as inputs to an 'OR' gate to generate this interrupt. The output of the comparator module enables the processor to replace the faulty module directly without trial and error. Therefore, in this process data is lost only during the switching period of the faulty main module. This switching period can be called fault-clearance period in this scheme. This fault-clearance period (*FcP*) comprises mainly three components as shown in Fig. 7.12 and by the following relation:

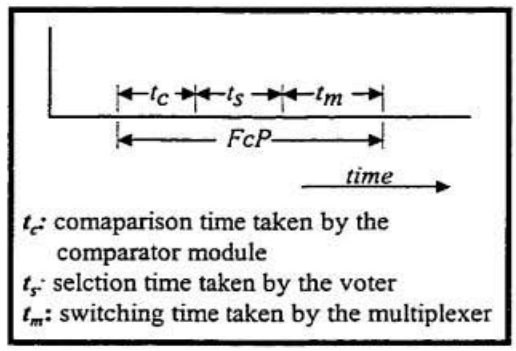


Figure 7.12: Fault-clearance time consists of three components.

$$FcP = t_c + t_s + t_m \tag{7.4}$$

As all these three operations are done in hardware, *FcP* is very small. As a result very few samples (one or two) are recoverable by using simple linear interpolation.

Due to the addition of extra hardware components to realize the voter, the reliability of the system decreases. This situation can be improved by implementing the voter in the software level provided that the software module does not contain bugs. Based upon the characteristics of particular application the designer will choose the specific approach for the implementation of the voter.

7.4.2 Restoration Using Software Implementation of Voting Algorithm with Triple Modular Redundancy

The software level implementation of the voter increases the value of t_S, which increases the length of the fault-clearance period. This increase imposes constraints on the restoration of lost samples using different restoration algorithms. Because the quality of the performance of available algorithms depends on the number of the lost samples (i.e., *FcP*) and the known signal characteristics, steps are taken to overcome this problem.

This problem is overcome by reading data from two channels simultaneously, similar to the case of dual redundancy. The signal from the support module is only used by the comparator to assist the software voter to detect the faulty module without trial and error. This scheme is better than dual redundancy in the sense that the chance of switching of the fault-free module will be avoided. Therefore, there will be no undefined samples in

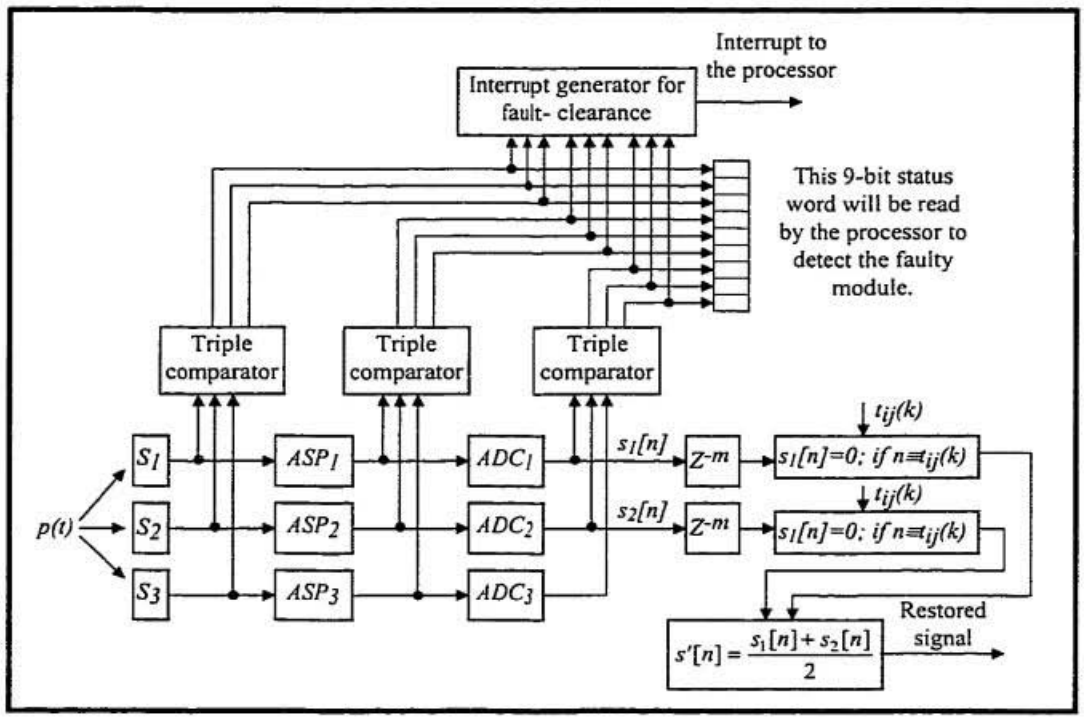


Figure 7.13: Restoration of signal in triple modular redundancy with software implementation of voting algorithm.

the $l_{ir}[u]$ periods. The overall system diagram is shown in Fig. 7.13. Due to the avoidance of the chance of switching of fault-free module this scheme has the potential to recover the signal completely. It should also be noted that this scheme outperforms the hardware voter through complete elimination of the need of interpolation to restore undefined samples. This scheme has been applied successfully in a laboratory set up to realize triple-modular redundant fault tolerant optical sensor as reported in Appendix G.

7.5 Generalized Fault-Tolerance Scheme

From the system development point of view, it is logical to have the provision of different levels of redundancy in the same system. This will maximize the use of the system resources. The system uses triple redundancy when there are three or more than three similar blocks. Dual modular redundancy is used when only two similar modules are available. When no redundant modules are available, the system uses the estimation algorithm to detect the fault. To achieve this objective it is necessary to have a switching module which allows both single and broadcast type of connections, such as the crossbar switch. The basic concept of a crossbar switch is shown in Fig. 7.14 [41]. Through this switch module any module in the input can be connected with any one in the output. It is also possible to connect an input module to multiple output modules. In Fig. 7.12., the input module, I_2 has been connected to O_1, and I_1 has been connected with both O_2 and O_n. This broadcast feature of crossbar allows different levels of redundancy in the same design. The overall system diagram is shown in Fig. 7.15.

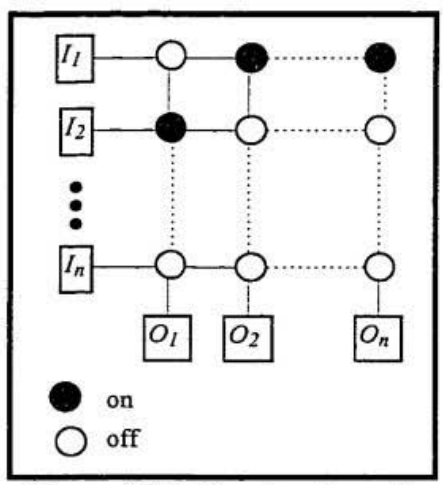


Figure 7.14: The structure of crossbar switch to connect the modules

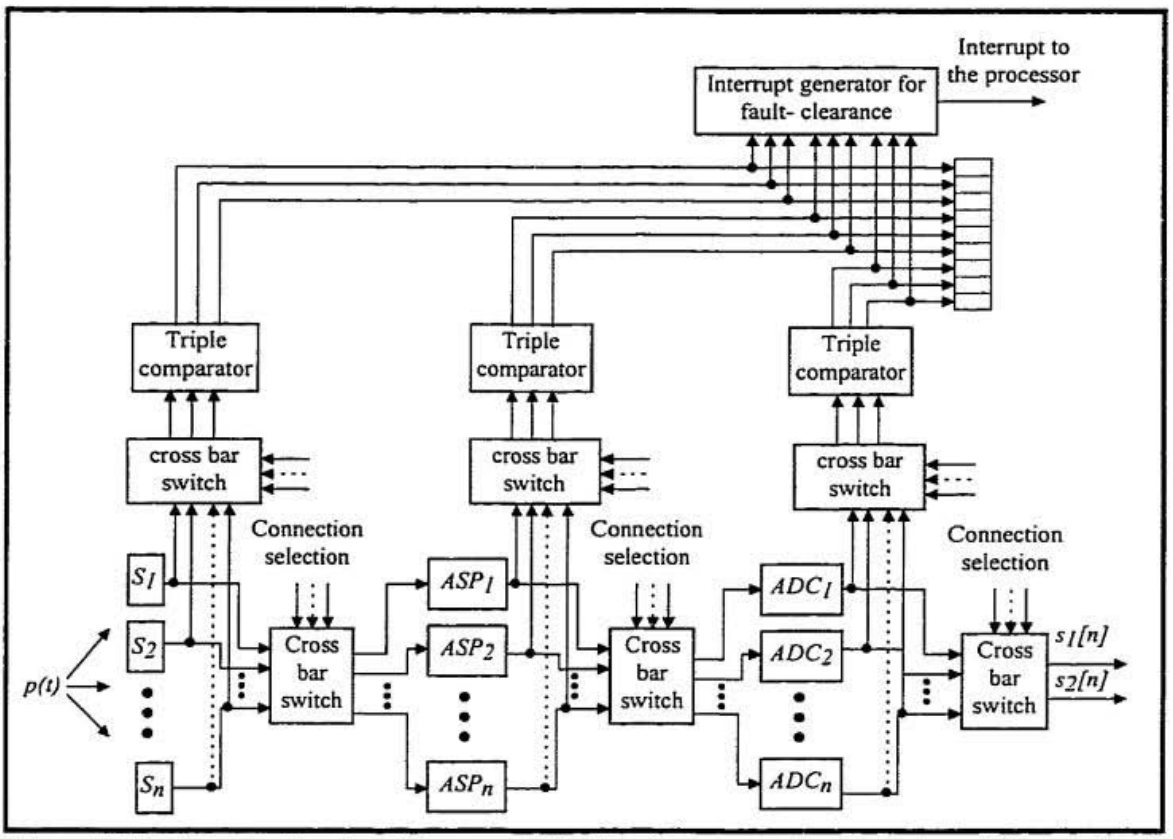


Figure 7.15: A generalized architecture of fault-tolerant sensing to achieve different level of redundancy.

For dual redundancy, the same module is connected to the two inputs of the comparator. In the case of no redundancy, the same module is connected to all three inputs of the comparator. The software maintains the record of the failed components and the level of redundancy.

7.6 Chapter Summary

The proposed scheme is not based on the recovery of lost samples from the information of the neighboring samples and is not limited to any class of signals (e.g., band limited signal). It does not require prior information about the signal characteristics. This scheme is computationally simple and is not iterative in nature. Therefore, this proposed scheme based on parallel sensing is fast and is capable of recovering lost samples of any class of signal. The addition of error to recover lost samples during switching periods through interpolation is low due to the availability of very fast switching devices. Although the requirements of hardware and software for the implementation of this scheme increase the system complexity, the rapid recovery of lost samples makes this scheme a useful solution for restoration of time sensitive signal in safety and mission critical operations.

Chapter 8

Conclusions and Recommendations for Future Work

8.1 Conclusions

The objective of this thesis work was to synthesize a novel engineering methodology for developing highly reliable sensor fusion systems of multi-sensori intelligent systems for the applications in the safety and mission critical environments. This methodology includes both the avoidance of faults during the development phase and the tolerance of sensor failures during the operation phase. The salient features of this thesis work are summarized in the following points:

1. Petri net based a novel discrete event framework has been developed to model requirements of sensor fusion systems as finite state machine. This framework allows the modeling of different modes of data integration: competitive, complementary, independent, and temporal in a unified manner. This framework has both the graphical and mathematical attributes. The intuitive graphical attribute has the potential facilitating communication between the developers and the clients to capture sensing requirements resulting in avoidance of requirement errors. The mathematical attribute will enable the developer to ensure logical and temporal correctness of the sensing requirements through the simulation of the modeled sensor fusion system. The effectiveness of this novel development has been demonstrated by simulating an example sensor fusion system.

2. A novel methodology of deriving the computing component specifications from the discrete event requirements (DEVR) model has been developed. This is based on the decomposition of DEVR model as hierarchical finite state machine. The use of the same formalism at different levels of sensing system decomposition will help avoid errors in deriving specifications of the computing components from client's high level sensing requirements. The use of the optimization technique in deriving temporal specifications of the computing components has been shown in order to enable the developer to optimize the cost of the underlying computing hardware. The use of this methodology has been illustrated by deriving the specifications of the computing components of an example sensor fusion system.

3. To ensure the temporal correctness of the sensor fusion system during the operation phase, the reasoning basis to derive the architecture of the underlying computing system from the sensing requirements has been developed.

4. A novel methodology to include redundant sensors to tolerate the failure of sensors during operation phase has been developed. The sensor fault tolerance using redundancy has been experimentally verified. The temporal overhead in incorporation of redundant sensors has been detected and necessary techniques have been developed to deal with this overhead. A fault-tree based novel technique to measure the dependence of different levels of fusion on the reliability of sensors has been developed. This technique has been used to derive the fault trees and reliability profiles of different levels of fusion of an example sensor fusion system.

5. The voting technique based fault detection scheme cannot detect transient faults generated due to the switching actions of the neighboring inductive loads (e.g., electric motors, electromagnetic relays) or due to electrostatic discharge in space and industrial environments. A computationally simple novel technique has been developed to detect the presence of transients in sensor data stream using local statistics.

6. The loss of sensor data during fault-clearance interval is one of the limitations of using voting technique based on redundancy for sensor fault tolerance. A novel parallel sensing based technique has been developed to address this problem. The implementation of this technique for fault tolerant sensor system of different levels of redundancy has been shown. The utility of this technique to restore data during fault-clearance interval for a triple modular optical sensor system has been experimentally verified.

The use of this formal, graphical, and mathematical technique will help the developer to avoid faults during the development phase. The use of redundancy will help tolerate sensor faults during the operation phase. The novel engineering methodology that is reported in this thesis has addressed different issues of fault avoidance and fault tolerance of sensor fusion systems in a unified framework. Therefore, it's the understanding of the author that this novel engineering methodology will enable the developer to engineer highly reliable sensor fusion systems of multi-sensori intelligent systems for the applications in safety and mission critical environments.

8.2 Recommendations for Future Works

The novel contributions reported in this thesis are the outcomes of a research work towards the development of a software system to automate the development process of highly reliable sensor fusion systems. Results of simulations and experiments have demonstrated the utilities of these contributions. There is a need to undertake development work to develop a software system in order to make the novel engineering methodology reported in this thesis readily usable by the development engineers.

It is necessary to develop a set of integrated discrete event software tools. This tool set will allow the modeling of requirements of sensor fusion system (SFS) as discrete event dynamic system. User friendly intuitive graphical user interface should be provided in order to facilitate communication between the developers and users. The simulation of this model will help ensure logical and temporal correctness of the SFS. This tool will also measure different features of the modeled SFS (e.g., the sensitiveness, the utilization of the operating time). This tool set will allow the decomposition of the DEVR model as hierarchical finite state machine in order to derive the specifications of the computing components. The architecture of the underlying computing system will also be derived with the help of this tool set in order to ensure the temporal correctness during run time. This same tool set will also generate control signals in order to execute the computing components in run time as an interpretation of modeled SFS. The fault trees and reliability profiles at different levels of fusion of sensor fusion systems will also be derived with the help of this tool set.

A repository of the computing components required for fusion of data should be developed. The discrete event tool set in run time will activate these components interpreting the discrete event model of sensor fusion systems. The use of same tool set in both development and operation phases will help the developer to avoid faults in realizing sensor fusion systems. This formalism will also be suitable for the enhancement of the features of already developed sensor fusion systems using this framework.

Due to the availability of high performance computing system and graphics library (e.g., OpenGL) at a reasonable price, the development work can be undertaken for the visualization of modeled sensor fusion systems as discrete event dynamic systems in 3-D graphics environment. This development work will include the visualization of the sensing systems including sensors and interactions among computing components, sensing environments, and sensed information. The computing components will process data generated by simulated sensors. This visualization scheme will further enhance the communication between the developers and clients resulting in better understanding of sensing requirements and limitations of different sensing schemes.

Further development work is required to validate the proposed local statistics based technique for the detection of transient faults present on sensor data stream. The development work should include the acquisition of transients generated by different phenomena (e.g., switching of inductive loads, electrostatic discharge in space, industrial, and laboratory environments) in wide variety of conditions, the modeling of these transients and the improvement of this proposed technique to make it capable for detecting.

In this thesis it has been shown that redundancy has the potential to improve the reliability of sensing system. In order to make this concept readily usable, it's necessary to undertake development work to develop fault-tolerant sensor modules using different levels of redundancy, so that SFS developer can use them as building blocks. These fault-tolerant sensor modules should implement the proposed parallel sensing based technique to recover sensor data lost during fault clearance interval. These fault-tolerant sensor modules should be smart enough to inform the higher level system modules about the status of different sensors, so that the system can avoid the interpretation of data provided by potentially faulty sensors.

The use of this methodology to develop complex sensor fusion systems will provide feedback in order to make this framework more effective and versatile. Moreover, there will be a need of continuous development of this framework to keep pace with the ever-increasing requirements of more complex intelligent systems.

References:

[1] J.L. Peterson, *Petri Net Theory and the Modeling of Systems,* Prentice-Hall, Inc., 1981.

[2] David L. Hall and James Llinas, "An Introduction to Multisensor Data Fusion," *Proceedings of the IEEE*, vol. 85, no. 1, January 1997, pp.6-23.

[3] David L. Hall, *Mathematical Techniques in Multisensory Data Fusion,* Artech House, Inc.1992.

[4] M. Mongi A. Abidi, and R.C. Gonzalez, *Data Fusion in Robotics and Machine Intelligence*, Academic Press, Inc. , 1992.

[5] B. V. Dasarathy, "Sensor Fusion Potential Exploitation - Innovative Architectures and Illustrative Applications," *Proceedings of the IEEE*, vol. 85, no. 1, January 1997, pp.24-38.

[6] S.S. Iyengar, L. Prasad, and Hla Min, *Advances in Distributed Sensor Integration*, pp. 65-81, Prentice Hall P T R, 1995.

[7] B.V. Dasarathy, *Decision Fusion*, IEEE Computer Society Press, 1994.

[8] N.G. Leveson, M.P. Erik, H. Hildreth, and J. D. Reese, "Requirements specification for process-control systems," *Proceedings of the IEEE*, vol. 82, no. 1, January 1994, pp. 684-707.

[9] J. M. Atlee and J. Gannon, "State-based model checking of event-driven system requirements," *IEEE Transactions on Software Engineering*, vol.19, no.1, January 1993, pp. 24-40.

[10] A. P. Ravn, H. Rischel, and K.M. Hansen, "Specifying and verifying requirements of real-time systems," *IEEE Transactions on Software Engineering*, vol.19, no. 1, January 1993, pp. 41-55.

[11] J.F. Meyer and H. Pham, "Fault-tolerant software: Guest editor's prolog," *IEEE Transactions on reliability*, vol. 42, no. 2, June 1993, pp. 117-118.

[12] A. Avizienis, "The N-version approach to fault-tolerant software," *IEEE Transaction on Software Engineering*, vol. SE-11, no. 12, December 1985, pp. 1491-1501.

[13] T.J. Shimeall and N. G. Leveson, "An empirical comparison of software fault tolerance and fault elimination," *IEEE Transactions on Software Engineering,* vol. 17, no. 2, February 1991, pp. 173-182.

[14] R. J. Abbott, "Resourceful systems for fault tolerance, reliability, and safety," *Computing Surveys,* vol. 22, no. 1, March 1990, pp. 35-68.

[15] Nancy G. Leverson, "Software Safety: Why, What, and How," *Computing Surveys*, vol. 18, no. 2, June 1986, pp.125-163.

[16] A. Gaskell and P. Probert, "Sensor Models and a Framework for Sensor Management," *Proceedings of SPIE-The International Society for Optical Engineering*, vol. 2059, pp. 2-13, 1993.

[17] M.E. Liggins II, C. Chong, I. Kadar, M.G. Alford, V. Vannicola, and S. Thomopoulos, "Distributed Fusion Architecture and Algorithms for Target Tracking," *Proceedings of the IEEE*, vol. 85, no. 1, January 1997, pp.95-107.

[18] S.S. Blackman and T.J. Broida, "Multiple Sensor Data Association and Fusion in Aerospace Applications," *Journal of Robotic Systems*, vol. 7, no. 3. 1990, pp. 445-485.

[19] Ren C. Luo and Michael G. Kay, "Multisensor Integration and Fusion in Intelligent Systems," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 19, no. 5, 1989, pp. 901-931.

[20] M. Nashman, B. Yoshimi, T.H. hong, W.G. Rippey, and M. Herman, "A Unique Sensor Fusion System for Coordinate Measuring Machine Tasks," *Proceedings of SPIE on Sensor Fsuion and Decentralized Control in Autonomous Robotic Systems*, Vol. 3209, 1997, pp. 145-156.

[21] K.A. Korzeniowski and E Woods, "Generic Architecture for Real-Time Multisensor Fusion Tracking Algorithm Development and Evaluation," *Proceedings of SPIE on Sensor Fusion VII*, Vol. 2355, 1994, pp. 33-42.

[22] A. Akerman III, "Pyramid Techniques for Multisensor Fusion," *Proceedings of SPIE on Sensor Fusion V*, vol. 1828, 1992, pp.124-131.

[23] R.W. Geiger and J.T. Snell, "Interdisciplinary Multisensory Fusion: Design Lessons from Professional Architects," *Proceedings of SPIE on Sensor Fusion V*, vol. 1828. 1992, pp.132-143.

[24] C. Bridgewater, C. Barral, and M. McGrath, "Sensor Integration in a Behavior-Based Architecture," *Proceedings of SPIE on Sensor Fusion V: Control Paradigms and Data Structures,* vol. 1161, 1991, pp. 496-503.

[25] P. Greenway, "SKIDS Data Fusion Project," *Proceedings of SPIE on Sensor Fusion V: Control Paradigms and Data Structures,* vol. 1161, 1991, pp. 504-515.

[26] S. Lee, E. Zapata, and P.S. Schenker, "Interactive and Cooperative Sensing and Control for Advanced Teleoperation," *Proceedings of SPIE on Sensor Fusion V: Control Paradigms and Data Structures,* vol. 1161, 1991, pp.516-530.

[27] F. Martinerie, "Data Fusion and Tracking Using HMMs in a Distributed Sensor Network," *IEEE Transactions on Aerospace and Electronics Systems*, vol. 33, no. 1, 1997, pp.11-28.

[28] M. Dekhil and T.C. Henderson, "Instrumented Sensor System Architecture," *The International Journal of Robotics Research*, vol. 17, no. 4, April 1998, pp.402-417.

[29] Y.C. Tang and C.S. George Lee, "A Geometric Feature Relation Graph Formulation for Consistent Sensor Fusion," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 22, no. 1, 1992, pp. 115-129.

[30] S.S. Iyengar and L. Prasad, "A General Computational Framework for Distributed Sensing and Fault-Tolerant Sensor Integration", *IEEE Transactions on Systems, Man, and Cybernetics*, vol.25, no. 4, 1995, pp. 643-650.

[31] J. Xu and D.L. Parnas, "On satisfying timing constraints in hard-real-time systems," *IEEE Transactions on Software Engineering*, vol.19, no.1, January 1993, pp. 70-84.

[32] T. Murata, "Petri nets: Properties, analysis and applications," *Proceedings of the IEEE*, vol. 77, no. 1, April 1989, pp. 541-580.

[33] A. Caloini, G. Magnani, and M. Pezze, "A Technique for Designing Robotic Control Systems Based on Petri Nets," *IEEE Transactions on Control Systems Technology*, vol. 6, no. 1, 1998, pp.72-87.

[34] P.J. Haas and G.S. Shedler, "Stochastic Petri Net representation of discrete event simulators," *IEEE Transactions on Software Engineering*, vol. 15, no. 11, April 1989, pp. 381-393.

[35] M. Kamath and N. Viswanadham, "Application of Petri Net based models in the medelling and analysis of flexible manufacturing systems," *Proceedings of IEEE International Conference on Robotics and automation*, 1986, pp. 312-317.

[36] J.J.P. Tasi, S. J. Yang, and Y. Chang, "Timing constraints Petri Nets and their application to schedualibility analysis of real-time system specification," *IEEE Transactions on Software Engineering,* vol. 21, no. 1, January 1995, pp. 32-49.

[37] G. Chiola, M. A. Marsan, G. Balbo, and G. Conte, "Generalized stochastic Petri Nets: a definition at the Net level and its applications," *IEEE Transactions on Software Engineering*, vol.19, no. 2, February 1993, pp. 89-107.

[38] C.V. Ramamoorthy and G.S. Ho, "Performance evaluation of asynchronous concurrent systems using Petri nets," *IEEE Transactions on Software Engineering*, vol. SE-6, no. 5, September 1980, pp. 440-449.

[39] M. Felder, D. Mandrioli, and A. Morzenti, "Proving properties of real-time systems through logical specifications and Petri Net models," *IEEE Transactions on Software Engineering*, vol. 20, no. 2, February 1994, pp. 127-141.

[40] M. W. Maier, "Integrated modelling: A unified approach to system engineering," *J. Systems Software*, vol. 32, 1996, pp.101-119.

[41] R. Gerber, S. Hong, and M. Saksena, "Guaranteeing real-time requirements with resource-based calibration of periodic process," *IEEE Transactions on Software Engineering*, vol. 21, no. 7, July 1995, pp. 579-578.

[42] I. Lee, P. Bremond-Gregoire, and R. Gerber, "A process algebraic approach to the specification and analysis of resource-bound real-time systems," ," *Proceedings of the IEEE*, vol. 82, no. 1, January 1994, pp. 158-171.

[43] B. P. Zeigler, "DEVS representation of dynamic systems: event-based intelligent control," *Proceedings of the IEEE*, vol. 77, no. 1, January 1989, pp. 72-80.

[44] K.M. Inan and P.P. Varaiya, "Algebras of discrete event models," *Proceedings of the IEEE*, vol. 77, no. 1, January 1989, pp. 24-38.

[45] G. Cohen, P. Moller, J. Quadrat, and M. Viot, "Algebraic tools for the performance evaluation of discrete event systems," *Proceedings of the IEEE*, vol. 77, no. 1, January 1989, pp. 39-58.

[46] D. L. Kiskis and K. G. Shin, "SWSL: A synthetic workload specification language for real-time systems," *IEEE Transactions on Software Engineering*, vol. 20, no. 10, October 1994, pp. 798-811.

[47] P. Inverardi and A. L. Wolf, "Formal specification and analysis of software architectures using the chemical abstract machine model," *IEEE Transactions on Software Engineering*, vol. 21, no. 4, April 1995, pp. 373-386.

[48] C. Ausfelder, E. Castelain, and J. Gentina, "A method for hierarchical modeling of the command of flexible manufacturing systems," *IEEE Transactions on Systems, Man, and Cybernetic*, vol. 24, no.4, April 1994, pp.564-573.

[49] M. Notomi and T. Murata, "Hierarchical reachability graph of bound Petri nets for concurrent-software analysis," *IEEE Transactions on Software Engineering*, vol. 20, no. 5, May 1994, pp. 325-336.

[50] J. Xu, "Multiprocessor Scheduling of Processes with Release Times, Deadlines, Precedence, and Exclusion Relations," *IEEE Transactions on Software Engineering*, vol. 19, no. 2, February 1993, pp.139-155.

[51] J. Zhu, T. G. Lewis, W. Jackson, and R. L. Wilson, "Scheduling in hard real-time application," *IEEE Software*, May 1995, pp. 54-63.

[52] J.H. Lala and R.E. Harper, "Architectural principles for safety-critical real-time applications," *Proceedings of the IEEE*, vol. 82, no. 1, January 1994, pp. 25-54.

[53] A. E. Barbour and A.S. Wojcik, "A general, constructive approach to fault-tolerant design using redundancy," *IEEE Transaction on computers*, vol. 38, no. 1, January 1989, pp. 15-29.

[54] K.G. Shin and P. Ramanathan, "Real-time computing: A new discipline of Computer Science and Engineering," *Proceedings of the IEEE*, vol. 82, no. 1, January 1994, pp. 6-24.

[55] M. Rokonuzzaman and R.G. Gosine, "An intelligent sensor fusion architecture for autonomous microgravity experiments," *Proceedings of the SPIE's Conference Sensor Fusion and Distributed Robotic Agents*, vol. 2905, November 1996, pp.53-63.

[56] M. Rokonuzaman and R.G. Gosine, "Minimization of the effect of fault-clearance period in the fault-tolerant sensing of an intelligent system," *Proceedings of the SPIE's conference Sensor Fusion and Decentralized Control in Autonomous Robotics Systems*, October .

[57] B.W. W. Johnson, *Design and Analysis of Fault-Tolerant Digital Systems*, Addison-Wesley Publishing Company, 1989.

[58] R. Billstein, S. Libeskind, and J.W. Lott, *A Problem Solving Approach to Mathematics*, The Benjamin/Cummings Publishing Company, Inc., 1990.

[59] R. James Firby, "Task directed sensing," *SPIE vol. 1198 Sensor fusion II: Human Machine Strategies* (1989), pp. 480-489.

[60] G. Schweizer, "Foundations for the ECBS Process," Proceedings of the ECBS'96, *Int'l IEEE Symposium and Workshop on Engineering of Computer-Based Systems*, 1996, pp. 16-22.

[61] M. Schwartz, *Telecommunication Networks: Protocols, Modeling and Analysis*, Addision-Wesley, 1987.

[62] Kai Hwang, *Advanced Computer Architecture: Parallelism, Scalability, Programmability*, McGraw-Hill, Inc., 1993.

[63] D. A. Patterson and J.L. Hennessy, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann Publishers, Inc., 1996.

[64] H. S. Stone, *High Performance Computer Architecture, Addision-Wesley*, 1993.

[65] C. Sundararajan, *Guide to Reliability Engineering: Data, Analysis, Applications, Implementation, and Management*, Van Nostrand Reinhold, 1991.

[66] J. L. Melsa and D.L. Cohn, *Decision and Estimation Theory*, McGraw-Hill, Inc., 1978.

[67] J.J. Goedbloed, *Electromagnetic Compatibility*, Prentice Hall, 1995.

[68] S.C. Lee, "Sensor Value Validation Based on Systematic Exploration of the Sensor Redundancy for Fault Diagnosis KBS," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 24, no. 4, 1994,pp 594-605.

[69] G. Rizzoni and Paul S. Min, "Detection of Sensor Failures in Automotive Engines," *IEEE Transactions on Vehicular Technology*, vol. 40, no. 2, 1991, pp.487-500.

[70] R.N. Clark and W. Setzer, "Sensor Fault Detection in a System with Random Disturbances," *IEEE Transactions on Aerospace and Electronic Systems*, vol. AES-16, no. 4, 1980, pp. 468-473.

[71] T. E. Menke, "Sensor/Actuator Failure Detecttion in the Vista F-16 by Multiple Model Adaptive Estimation," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 31, no. 4, 1995, pp.1218-1228.

[72] P. Hsu, K. Lin, and L. Shen, "Diagnosis of Multiple Sensor and Actuator Failures in Automotive Engines," *IEEE Transactions on Vehicular Technology*, vol. 44, no. 4, 1995, pp. 779-789.

[73] N. P. Piercy, "Sensor Failure Estimators for Detection Filters," *IEEE Transactions on Automatic Control*, vol. 37, no. 10, 1992, pp. 1553-795.

[74] A.S. Willsky, "A Survey of Design Methods for Failure Detection in Dynamic Systems," *Automatica*, vol. 12, pp. 601-611.

[75] J. C. Deckert, M.N. Desai, J.J. Deyst and A. S. Willsky, "F-8 BFBW Sensor Failure Identification Using Analytical Redundancy," *IEEE Transactions on Automatic Control*, vol. AC-22, no. 5, 1977, pp. 795-803.

[76] Y. Maki and K.A. Loparo, "A Neural-Network Approach to Fault Detection and Diagnosis in Industrial Process," *IEEE Transactions on Control System Technology*, vol. 5, no. 6, 1997, pp. 529-541.

[77] R. Dorr, F. Kratz, J. Ragot, F. Loisy, and J. Germain, "Detection, Isolation, and Identification of Sensor Faults in Nuclear Power Plants," *IEEE Transactions on Control System Technology*, vol.5, no. 1, 1997, pp.42-60.

[78] S. Murugesan and P.S. Goel, "A Scheme for Fault Tolerance in Earth Sensors," *IEEE Transactions on Aerospace and Electronic Systems*, vol. AES-25, no. 1, 1989, pp. 21-29.

[79] M. Rokonuzzaman and R.G. Gosine, "Adaptive Fuzzy_Statistical Decision Model to Grade Sensor Data," *Proceedings of the IEEE Canadian Conference on Elctrical and Computer Engineering*, vol. II, 1997, pp. 773-776.

[80] M. Rokonuzzaman and R.G. Gosine, "Fault-tolerant Sensor Fusion Architecture for Mission Critical Applications", *The Proceedings of the Seventh Annual Newfoundland Electrical and Computer Engineering Conference*, May 3, 1996.

[81] R. Veldhuis, *Restoration of Lost Samples in Digital Signals*, Prentice Hall International (UK) Ltd.,1990,

[82] R.J. Marks II, "Restoring lost samples from an oversampled bandlimited signal", *IEEE Transaction on ASSP*, vol. 31, no.3, pp. 752-755, 1983.

[83] N. Erdol, C. Castelluccia and A. Zilouchian, "Recovery of Missing Speech Packets Using the Short-Time Energy and Zero-Crossing Measurements," *IEEE Transaction on Speech and Audio Processing*, vol. 1, no. 3, pp.295-313, 1993.

[84] L. L. Scharf, *Statistical Signal Processing: Detection, Estimation, and Time Series Analysis*, Addison-Wesley Publishing Company, 1991.

[85] S. J. Godsill and Peter J.W. Rayner, "A Bayesian Approach to the Restoration of Degraded Audio Signals," *IEEE Transaction on Speech and Audio Processing*, vol. 3, no. 4, pp.267-277, 1995.

[86] J. Feng, K. Lo, and H. Mehrpour, "Error Concealment for MPEG Video Transmissions," *IEEE Transaction on Consumer Electronics*, vol. 43, no. 2, pp. 183-186, 1997.

[87] Y. Chen and B. Chen, "Model-based Multirate Representation of Speech Signals and its Application to Recovery of Missing Speech Packets," *IEEE Transaction on Speech and Audio Processing*, vol. 5, no. 3, pp.220-231, 1997.

Appendix A

A Design Problem to Verify the Discrete Event Framework to Engineer a Reliable Sensor Fusion System

A.1 Introduction

This is an example design problem to verify the different proposed modeling techniques and algorithms developed in this thesis. The parameters of this design problem have been chosen to demonstrate the capability of the proposed discrete event framework to deal with worst case sensing scenarios. This design problem is stated in the following section.

A.2 Problem Statement

1. The set of sensors is, $\mathbf{S}=\{S_1, S_2, \ldots, S_7\}$.
2. The sensing sequence is defined in the Table A.1 in terms of phases and periods.

Table A.1: The phases and periods of sensing

Sensors	**Range of phases (in units)**		**Range of periods (in units)**	
	Φ^l :Lower limit	Φ^u :Upper limit	T^l :Lower limit	T^u :Upper limit
S_1	0	0	1900	2000
S_2	50	100	1800	2000
S_3	200	250	2000	3000
S_4	300	400	2000	2000
S_5	500	550	3000	3000
S_6	600	650	2000	3000
S_7	800	850	1000	1100

3. The set of conditions is, **PC**={Pc_1, Pc_2, . . . , Pc_{12}}. The lifetimes of these conditions are shown in Table A.2.

Table A.2: The lifetimes (in units) of the conditions

	The conditions											
	Pc_1	Pc_2	Pc_3	Pc_4	Pc_5	Pc_6	Pc_7	Pc_8	Pc_9	Pc_{10}	Pc_{11}	Pc_{12}
Life	25	27	28	29	100	36	19	48	35	20	25	30

4. The set of sensors specific periodic processes (activities) is, **PE**={PE_1, PE_2, . . . , PE_7}. The service times of these processes are shown in Table A.3.

Table A.3: The service times (in units) of the periodic processes

	Periodic processes						
Service times	PE_1	PE_2	PE_3	PE_4	PE_5	PE_6	PE_7
Lower limit	10	18	12	14	16	20	25
Upper limit	15	22	14	16	19	27	30

5. The set of aperiodic or conditional processes (activities) is, **AE**={AE_1, AE_2, . . ., AE_8}. The service times of these processes are shown in Table A.4.

Table A.4: The service times (in units) of aperiodic processes

	Aperiodic processes							
Service times	AE_1	AE_2	AE_3	AE_4	AE_5	AE_6	AE_7	AE_8
Lower limit	12	14	15	16	20	18	12	15
Upper limit	16	16	19	20	24	20	14	16

6. The relationships between conditions and processes are shown in the Tables A.5-A.7.

Table A.5: Generation of maximum number of conditions (tokens) by periodic processes

	The periodic processes						
The conditions	**PE_1**	**PE_2**	**PE_3**	**PE_4**	**PE_5**	**PE_6**	**PE_7**
Pc_1	1	1	0	0	0	0	0
Pc_2	0	1	0	0	0	0	0
Pc_3	0	1	1	0	0	0	0
Pc_4	0	0	0	1	1	0	0
Pc_5	0	0	0	0	1	0	0
Pc_6	0	0	0	0	0	1	0
Pc_7	0	0	0	0	0	0	1
Pc_8	0	0	0	0	0	0	0
Pc_9	0	0	0	0	0	0	0
Pc_{10}	0	0	0	0	0	0	0
Pc_{11}	0	0	0	0	0	0	0
Ps_1	Ts_1	Ts_2	Ts_3	Ts_4	Ts_5	Ts_6	Ts_7

Table A.6:Generation of maximum number of conditions (tokens) by aperiodic processes

	The aperiodic processes							
The conditions	**AE_1**	**AE_2**	**AE_3**	**AE_4**	**AE_5**	**AE_6**	**AE_7**	**AE_8**
Pc_1	0	0	0	0	0	0	0	0
Pc_2	0	0	0	0	0	0	0	0
Pc_3	0	0	0	0	0	0	0	0
Pc_4	0	0	0	0	0	0	0	0
Pc_5	0	0	0	0	0	0	0	0
Pc_6	0	0	0	0	0	0	0	0
Pc_7	0	0	0	0	0	0	0	0
Pc_8	0	1	1	0	0	0	0	0
Pc_9	0	0	0	1	0	0	0	0
Pc_{10}	0	0	0	1	1	0	0	0
Pc_{11}	0	0	0	0	0	1	1	0
Pd	1	0	0	0	1	0	0	1

Table A.7: Absorption of conditions by aperiodic processes

	The aperiodic processes							
The conditions	**AE_1**	**AE_2**	**AE_3**	**AE_4**	**AE_5**	**AE_6**	**AE_7**	**AE_8**
Pc_1	1	0	0	1	0	0	0	0
Pc_2	1	1	1	0	0	0	0	0
Pc_3	0	0	0	1	0	0	0	0
Pc_4	0	1	0	0	1	0	0	0
Pc_5	0	0	0	1	0	0	0	0
Pc_6	0	0	0	0	1	0	0	0
Pc_7	0	0	0	0	1	0	0	0
Pc_8	0	0	0	0	0	1	0	0
Pc_9	0	0	0	0	0	1	0	0
Pc_{10}	0	0	0	0	0	0	1	0
Pc_{11}	0	0	0	0	0	0	0	1
Ps_1	0	0	0	0	0	0	0	0

An event (periodic or aperiodic) will generate conditions by placing tokens in particular places. For example, aperiodic event AE_2 can generate the condition Pc_8 by placing a token in place Pc_8. An event will absorb conditions by removing token from places. For example, the event AE2 will absorb tokens from places Pc_2 and Pc_4. An event may or may not generate conditions (tokens in places). The generation of conditions depends upon the state of sensing environment (e.g., the periodic process, sensing temperature sensor, may generate a condition if temperature goes beyond certain threshold value, the specific value depends upon the sensing requirements of a particular environment).

These tables represent the discrete requirements of customers in a summarized form. The information in this tabular form will be generated in the requirements definition phase of the system development. In this phase, each discrete requirement (e.g., the execution of a particular aperiodic process) will be treated as a separate entity. This approach will enable the system developer to capture the of user's requirements with fewer errors.

Appendix B

[redacted] Verification of Discrete Event Requirements Model of SFS by Simulation

B.1 Introduction

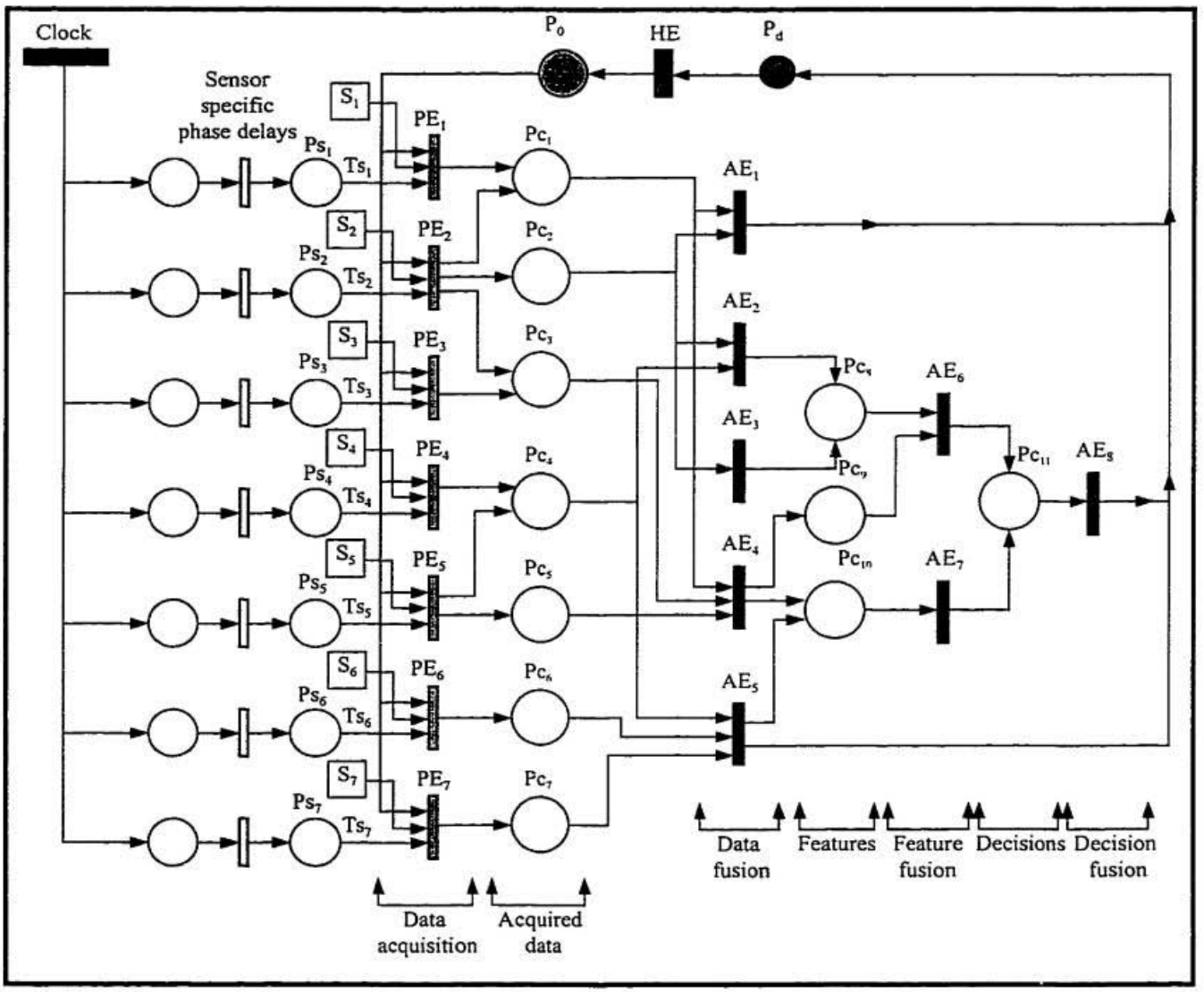


Figure B.1: The Petri net model of the specified example sensor fusion system

The DEVR model of the sensor fusion system as specified in Appendix A is shown in Fig.B.1. In this modeled SFS, sensor data are acquired in a particular sequence and are integrated exploiting temporal relationships among these sensed data. In this integration process, sensor data are integrated in different modes. For example, sensors 1 and 2 are playing redundant roles due to the placement of tokens by periodic events PE_1 and PE_2 at the same place Pc_1 and the requirement of only one token by the aperiodic event AE_1. The mode of data integration from these two sensors can be defined as complementary by adopting the constraint that AE_1 needs 2 tokens from the place Pc_1. Using the conventional terminology of sensor fusion the role of these events are shown in Table B.1.

Table B.1: Different levels of data integration in the example SFS

Data acquisition	Data fusion	Feature fusion	Decision fusion
PE_1			
PE_2	AE_1		
PE_3	AE_2	AE_6	
PE_4	AE_3		AE_8
PE_5	AE_4	AE_7	
PE_6	AE_5		
PE_7			

Here, in the data fusion level the decisions placed by the event AE_6 and AE_7 in place Pc_{11} play redundant role. But they can be complementary if AE_8 requires two decisions from the place Pc_{11}.

The input and output functions of the modeled SFS are defined by the following matrices D^- and D^+ respectively. To define the input function, the feedback from the reasoning sub-system to the periodic processes shown as dashed arrows have not been considered, because the reasoning sub-system is outside the scope of this work. To capture the interaction between the sensor specific periodic processes and aperiodic processes, the tokens generated by the clock and the phase related processes have not been considered to define the output function.

$$
D^- =
\begin{array}{c}
 \\
PE_1 \\
PE_2 \\
PE_3 \\
PE_4 \\
PE_5 \\
PE_6 \\
PE_7 \\
AE_1 \\
AE_2 \\
AE_3 \\
AE_4 \\
AE_5 \\
AE_6 \\
AE_7 \\
AE_8 \\
HE
\end{array}
\begin{array}{c}
\begin{array}{cccccccccccccccccccc}
Ps_1 & Ps_2 & Ps_3 & Ps_4 & Ps_5 & Ps_6 & Ps_7 & Pc_1 & Pc_2 & Pc_3 & Pc_4 & Pc_5 & Pc_6 & Pc_7 & Pc_8 & Pc_9 & Pc_{10} & Pc_{11} & Pd & P_o
\end{array} \\
\begin{bmatrix}
Ts_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & Ts_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & Ts_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & Ts_4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & Ts_5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & Ts_6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & Ts_7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{bmatrix}
\end{array}
$$

(B.1)

$$
D^+ =
\begin{array}{c}
 \\
PE_1 \\
PE_2 \\
PE_3 \\
PE_4 \\
PE_5 \\
PE_6 \\
PE_7 \\
AE_1 \\
AE_2 \\
AE_3 \\
AE_4 \\
AE_5 \\
AE_6 \\
AE_7 \\
AE_8 \\
HE
\end{array}
\begin{array}{c}
\begin{array}{cccccccccccccc}
Pc_1 & Pc_2 & Pc_3 & Pc_4 & Pc_5 & Pc_6 & Pc_7 & Pc_8 & Pc_9 & Pc_{10} & Pc_{11} & P_d & P_0
\end{array} \\
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}
\end{array}
$$

(B.2)

To study the salient features of the modeled sensor fusion system, the matrix D^- defining the input function can be partitioned into a number of sub-matrices. The first one is at the leftmost top corner as shown below.

$$D^-1 = \begin{bmatrix} Ts_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & Ts_2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & Ts_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & Ts_4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & Ts_5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & Ts_{61} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & Ts_7 \end{bmatrix} \quad \text{(B.3)}$$

According to this proposed framework, every sensor is periodically sensed with unique periodic process. Therefore, the number of periodic processes is equal to the number of sensors and the diagonal matrix represents the relationship among them. So, if this sub-matrix of a modeled SFS is other than a diagonal matrix, the system has not been modeled properly using this framework. For a system of n number of sensors, this sub-matrix is a n x n diagonal matrix; where the diagonal elements are the sensing periods of the corresponding sensors.

The second sub-matrix is at the left-most bottom one relating the places Ps_1,...Ps_7, AE_1,...,AE_8, and HE as shown below.

$$D^-2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \text{(B.4)}$$

This is a null matrix. According to this framework, the clock drives only the periodic processes; therefore aperiodic processes do not have any direct link with the clock. So, for a modeled SFS if this sub-matrix is other than null, the system has not been modeled correctly using this framework. For a system of n sensors, this sub-matrix is a matrix of n

column and any number of rows. The number of rows will depend upon the number of aperiodic events.

The third sub-matrix is at the top-most right corner in the input definition matrix D^-. This sub-matrix relates the periodic events $PE_1,...,PE_7$ with $Pc_1,...,Pc_{11}$, P_d, and P_0.

$$D^-3 = \begin{bmatrix} 0&0&0&0&0&0&0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&0&0&0&0&0&0&1 \end{bmatrix} \tag{B.5}$$

According to this modeling framework, tokens in the output places of periodic and aperiodic events, and house keeping event do not drive the periodic events. Only the token in the place P_0 drives the periodic events. Therefore, all the elements, except those of the right-most column of this sub-matrix, are zero and all the elements of the right most column are 1. If the matrix $D^-.3$ does not satisfy this criteria, the modeled system has a flaw according to this framework. If there are n number of sensors, p conditional places, this third sub-matrix dimension is (n x $p+2$).

The fourth sub-matrix defines the generation of all aperiodic events and this is at the right bottom part of the input definition matrix as shown below. Here, the rows are the aperiodic events $AE_1,...,AE_8$ and the columns are places for the conditions $Pc_1,...,Pc_{11}$.

$$D^-.4 = \begin{bmatrix} 1&1&0&0&0&0&0&0&0&0&0 \\ 0&1&0&1&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0&0&0&0 \\ 1&0&1&0&1&0&0&0&0&0&0 \\ 0&0&0&1&0&1&1&0&0&0&0 \\ 0&0&0&0&0&0&0&1&1&0&0 \\ 0&0&0&0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&0&0&0&1 \end{bmatrix} \tag{B.6}$$

The number of non-zero element in a column is the possible value of the token placed in that place. For example, if the token in place Pc_2 represents the size of a target, in this case the target may have three different sizes (e.g., small, medium, big). These different values or color of token (e.g., size of the target) are used in this model for branching to generate appropriate events to address the issue of detection of different sizes of targets. The presence of more than one number in a cell indicates that the corresponding place receives the same token from more than one process (e.g., periodic sensor sensing). These redundant tokens may be used by the corresponding event to increase reliability. If there are *m* number of aperiodic processes and *n* number of conditional places, the dimension of this matrix is a *m* x *n*.

The 5th sub-matrix relates aperiodic processes $AE_1,...,AE_8$, P_d with P_0 as shown in Eq.(B.7). This is a null matrix, because the tokens in the places P_d and P_0 do not generate aperiodic events according to this modeling formalism. The rows of this matrix represent aperiodic events and the columns correspond to P_d and P_0. If there are m aperiodic events, the dimension of this matrix is *m* x 2.

$$D^{-}5 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{B.7}$$

The input relationship of the house keeping event, HE, with the conditional places, $Pc_1,...Pc_{11}$, P_d and P_0 is shown by the sixth sub-matrix given in Eq.(B.8). All the elements of this matrix are zero except the second element from the right. If there are n conditional places, the dimension of the matrix is *1* x *n+2*.

$$D^{-}.6 = [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0] \tag{B.8}$$

The partitioning process for the input definition matrix can be repeated for the output definition matrix D^+ to reveal more features of the modeled system. The generation of the tokens in the places, $Pc_1,...,Pc_{11}$, by all periodic and aperiodic events, $PE_1,...,PE_7,AE_1,...,AE_8$, is shown by the top left-most sub-matrix of the output definition matrix as shown below. Here, the rows represent the events and the columns are places.

$$D^+1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \text{(B.9)}$$

Multiple non-zero elements in a particular column indicate that the corresponding events are playing redundant role by generating tokens in the same place. For a system of m events (both periodic and aperiodic) and n conditional places the size of this matrix is m x n. According to this modeling paradigm, the possible values of an element of this matrix are 0 or 1.

The terminal events are identified by the 2nd sub-matrix of the output definition matrix relating all periodic and aperiodic events with the place P_d as shown in Eq.(B.10). This matrix has been represented in transpose form.

$$D^+2 = [0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 1 \;\; 0 \;\; 0 \;\; 0 \;\; 1 \;\; 0 \;\; 0 \;\; 1]^T \quad \text{(B.10)}$$

The events corresponding to the non-zero elements of this matrix are terminal events. In this present example, the terminal events are AE_1, AE_8, AE_5. According to this design formalism, all the elements of this sub-matrix must not be zero.

The third one is a null matrix relating all periodic and aperiodic events with the place P_0 as shown below in transpose form.

$$D^+3 = [0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0]^T \qquad \text{(B.11)}$$

The forth sub-matrix relates the token generation of the housekeeping event, HE, with all conditional places, P_d and P_0 as shown in Eq.(B.12). According to this framework, all the elements of this sub-matrix are zero except the right most one.

$$D^+.4 = [0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 1] \qquad \text{(B.12)}$$

The check of the characteristics of these sub-matrixes for a particular modeled SFS will enable the designer to have a preliminary check to ensure that the system has been modeled according to this formalism. Due to the numerical nature of these verification techniques, software based automation is quiet feasible for this work. It's the author's understanding that this automation has the potential to improve the accuracy of the design and speed up the design process. Therefore, it is reasonable to conclude that this formalism will enable the designer to partially avoid faults at the very early stage of system development resulting in better reliability of system performance.

The execution path analysis will enable the designer to check logical and temporal correctness of the modeled system. The technique of this execution path analysis is depicted in the following sub-section.

B.2 Execution Path and Time Analysis

The value (i.e., color) of a token in a place is used by the SFS for branching decision. The execution of a process (i.e., service of an event) may generate tokens in more than one place resulting in parallel operation of multiple processes.

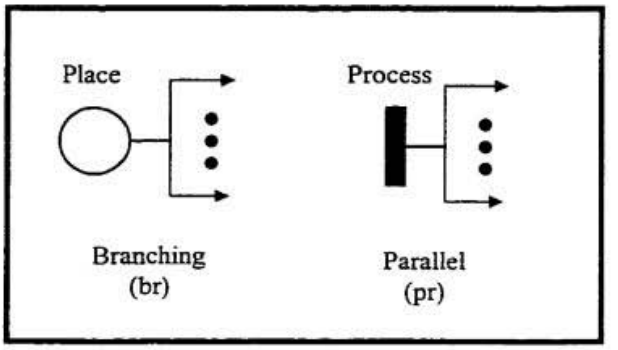


Figure B.2: The branching and parallel operations in the Petri net model of the SFS.

This concept is depicted in Fig.B.2. The underlying computing system may execute these parallel processes simultaneously or sequentially.

B.2.1 Execution Paths From the First Sensor

The sequence of execution of different processes to serve the sensing requirements of the 1st sensor is shown in Fig. B.3.

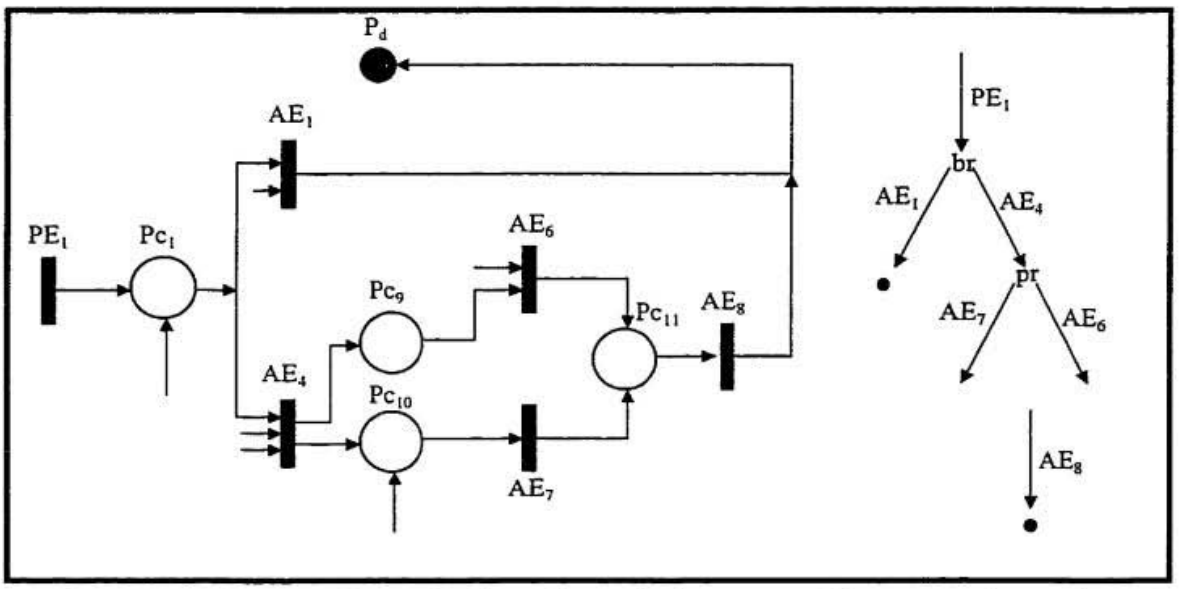


Figure B.3. The execution paths from the periodic process PE_1 to serve the sensing of the 1st sensor.

The execution paths to serve the first sensor are defined by the following equations

$$S_1.p_1=PE_1,AE_1 \quad (B.13)$$

$$S_1.p_2=PE_1,AE_4,AE_7,AE_8 \quad (B.14)$$

$$S_1.p_3=PE_1,AE_4,AE_6,AE_8 \quad (B.15)$$

The maximum total time to serve the event of periodic sensing of the first sensor is

$$S_1.t= \max\{ (PE_1.t+AE_1.t), (PE_1.t+AE_4.t+AE_7.t+ AE_6.t+AE_8.t)\} \quad (B.16)$$

for sequential execution of two parallelizable processes AE_6 and AE_7 , or

$$S_1.t= \max\{ (PE_1.t+AE_1.t), (PE_1.t+AE_4.t+\max.(AE_7.t, AE_6.t)+AE_8.t)\} \quad (B.17)$$

for parallel executions of AE_6 and AE_7.

B.2.2 Execution Paths from the Second Sensor

The execution paths for sensing the second sensor are shown in Fig.B.4.

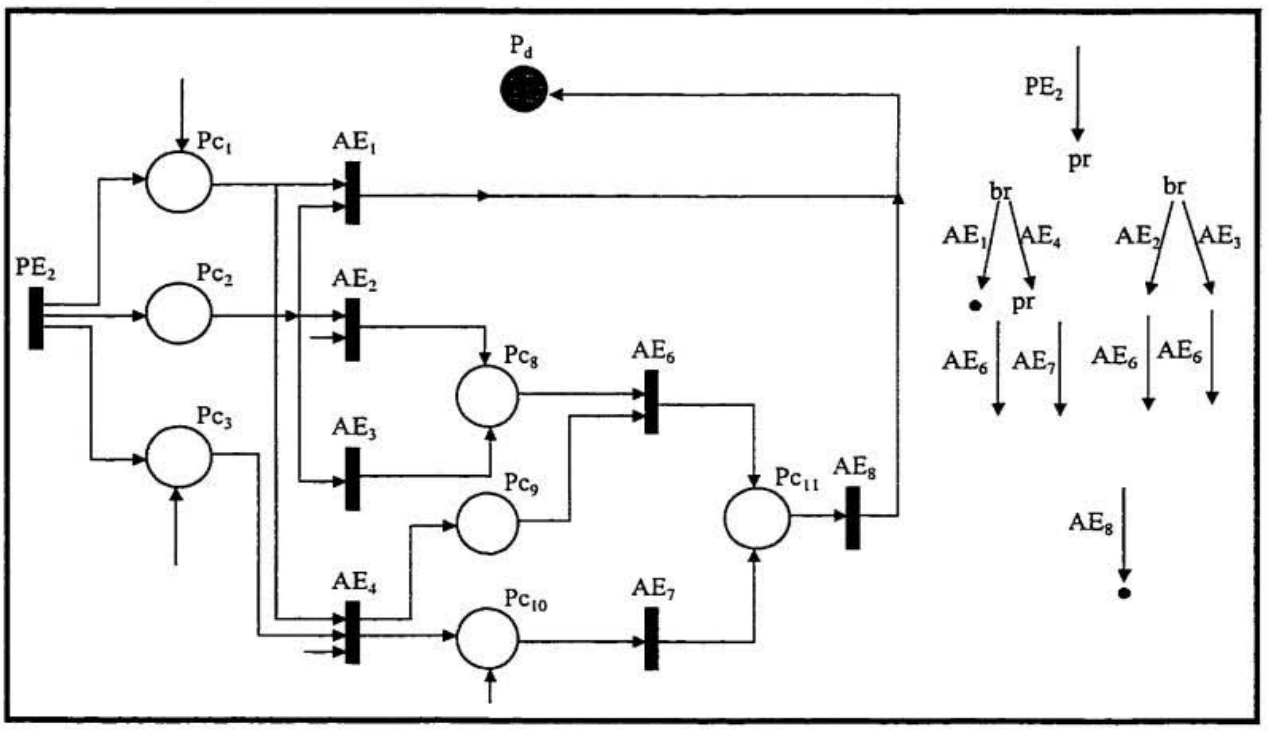


Figure B.4: The execution paths from the periodic process PE_2 to serve sensing of the second sensor.

The execution paths to serve the second sensor are defined by the following equations

$$S_2.p_1=PE_2, AE_1 \quad (B.18)$$

$$S_2.p_2=PE_2, AE_4, AE_6, AE_8 \quad (B.19)$$

$$S_2.p_3= PE_2, AE_4, AE_7, AE_8 \quad (B.20)$$

$$S_2.p_4=PE_2, AE_2, AE_6, AE_8 \quad (B.21)$$

$$S_2.p_5= PE_2, AE_3, AE_6, AE_8 \quad (B.22)$$

The maximum total time for periodic sensing of the second sensor is

$$S_2.t= PE_2.t+\max. \{(AE_1), (AE_4+AE_6+AE_7+AE_8)\}+\max. \{(AE_2+AE_6+AE_8), (AE_3+AE_6+AE_8)\} \quad (B.23)$$

for sequential execution of parallelizable processes, or

$$S_2.t= PE_2.t+\max. [\max.\{(AE_1), (AE_4+\max. (AE_6, AE_7)+AE_8)\}, \max. \{(AE_2+AE_6+AE_8), (AE_3+AE_6+AE_8)\}] \quad (B.24)$$

for parallel execution of parallelizable processes.

B.2.3 Execution Paths from the Third Sensor

The execution paths for sensing the third sensor are shown in Fig.B.5.

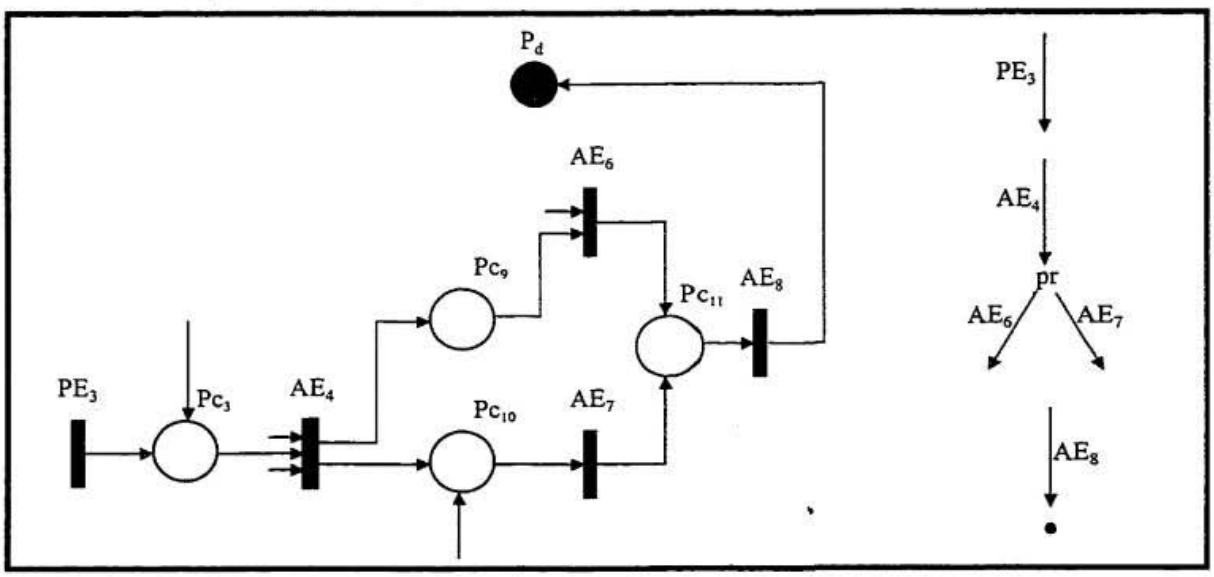


Figure B.5: The execution paths from the periodic process PE_3 to serve sensing of the third sensor.

The execution paths to serve the third sensor are defined by the following equations

$$S_3.p_1=PE_3, AE_4, AE_6, AE_8 \quad (B.25)$$

$$S_3.p_2=PE_3, AE_4, AE_7, AE_8 \quad (B.26)$$

The maximum total time for periodic sensing of the third sensor is

$$S_3.t= PE_3.t+ AE_4.t+ AE_6+AE_7.t+ AE_8.t \quad (B.27)$$

for sequential execution of parallelizable processes, or

$$S_3.t= PE_3.t+ AE_4.t+\max.(AE_6, AE_7.t)+ AE_8.t \quad (B.28)$$

for parallel execution of parallelizable processes.

B.2.4 Execution Paths from the Fourth Sensor

The execution paths for sensing the fourth sensor are shown in Fig.B.6.

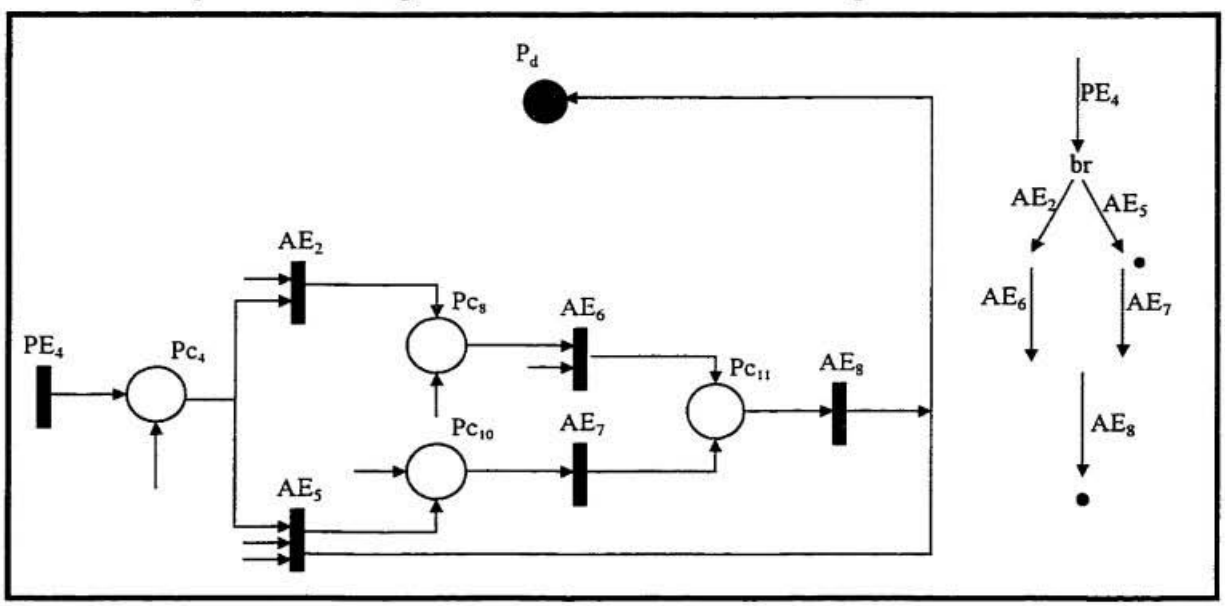


Figure B.6: The execution paths from the periodic process PE_4 to serve sensing of the fourth sensor.

The execution paths to serve the fourth sensor are defined by the following equations

$$S_4.p_1=PE_4, AE_2, AE_6, AE_8 \quad (B.29)$$

$$S_4.p_2=PE_4, AE_5 \quad (B.30)$$

$$S_4.p_3=PE_4, AE_5, AE_7, AE_8 \quad (B.31)$$

The maximum total time for periodic sensing of the fourth sensor is

$$S_4.t= PE_4.t+ \max. \{(AE_2.t+ AE_6.t+AE_8.t), AE_5.t, (AE_5.t+ AE_7.t+AE_8.t)\} \quad (B.32)$$

B.2.5 Execution Paths from the Fifth Sensor

The execution paths for sensing the fifth sensor are shown in Fig.B.8.

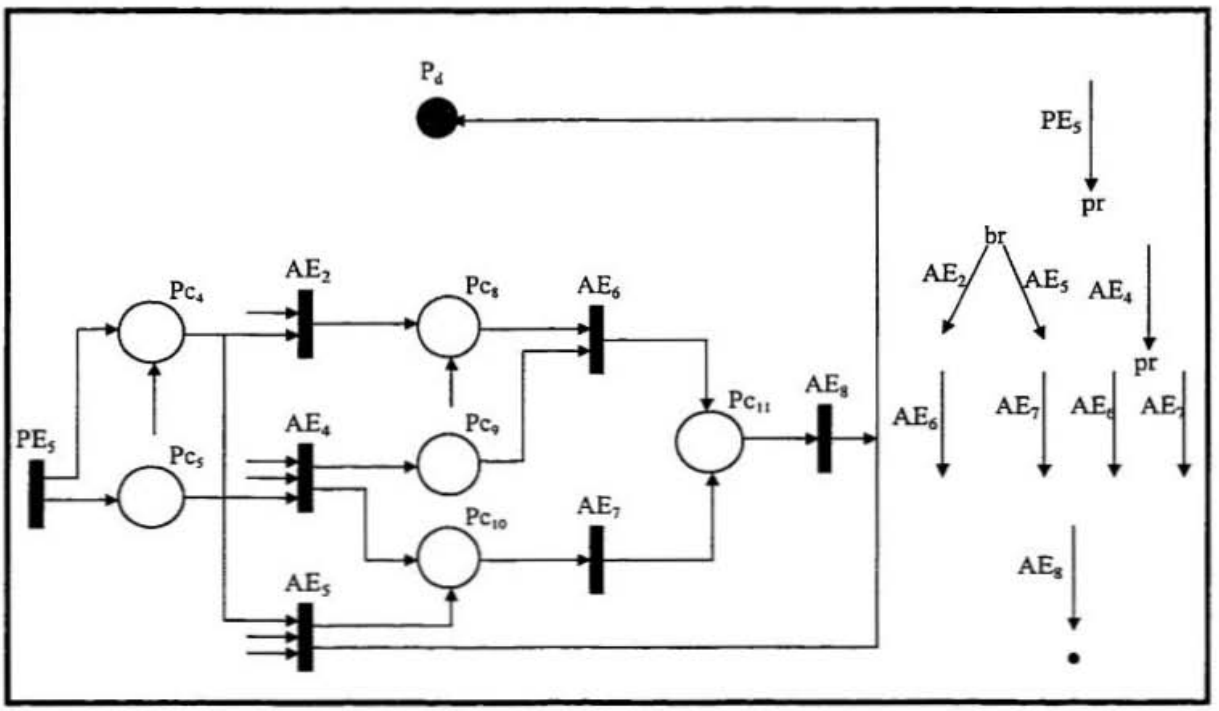


Figure B.8: The execution paths from the periodic process PE_5 to serve sensing of the fifth sensor.

The execution paths to serve the fifth sensor are defined by the following equations

$$S_5.p_1=PE_5, AE_2, AE_6, AE_8 \quad (B.33)$$

$$S_5.p_2=PE_5, AE_5, AE_7, AE_8 \quad (B.34)$$

$$S_5.p_3=PE_5, AE_4, AE_6, AE_8 \quad (B.35)$$

$$S_5.p_4=PE_5, AE_4, AE_7, AE_8 \quad (B.36)$$

The maximum total time for periodic sensing of the fifth sensor is

$$S_5.t = PE_5.t + [\max.\ \{(AE_2.t + AE_6.t), (AE_5.t + AE_7.t)\} + AE_4.t + (AE_6.t + AE_7.t)] + AE_8.t \quad (B.37)$$

for sequential execution of parallelizable processes, or

$$S_5.t = PE_5.t + \max.\ [\max.\{(AE_2.t + AE_6.t), (AE_5.t + AE_7.t)\},\ AE_4.t + \max.\ (AE_6.t, AE_7.t)] + AE_8.t \quad (B.38)$$

for parallel execution of parallelizable processes.

B.2.6 Execution Paths From the Sixth Sensor

The execution paths for sensing the sixth sensor is shown in Fig.B.9.

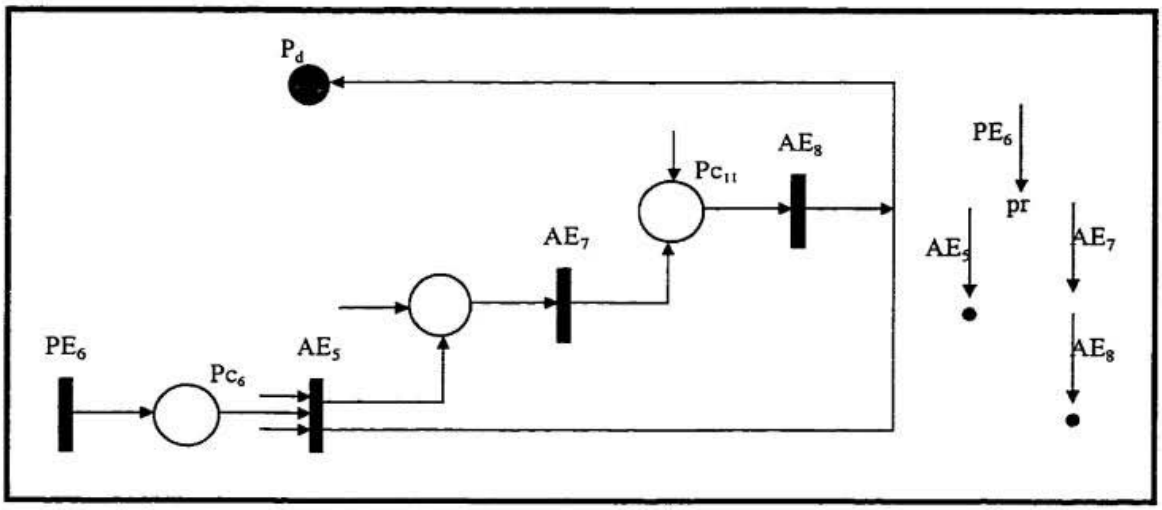


Figure B.9: The execution paths from the periodic process PE_6 to serve sensing of the sixth sensor.

The execution paths to serve the sixth sensor are defined by the following equations

$$S_6.p_1 = PE_6, AE_5 \quad (B.39)$$
$$S_6.p_2 = PE_6, AE_7, AE_8 \quad (B.40)$$

The maximum total time for periodic sensing of the sixth sensor is

$$S_6.t = PE_6.t + (AE_5.t + AE_7.t + AE_8.t) \quad (B.41)$$

for sequential execution of parallelizable processes, or

$$S_6.t = PE_6.t + \max.\ \{\ AE_5.t,\ (AE_7.t + AE_8.t)\ \} \qquad (B.42)$$

for parallel execution of parallelizable processes.

B.2.7 Execution Paths from the Seventh Sensor

The execution paths for sensing the seventh sensor are shown in Fig.B.10.

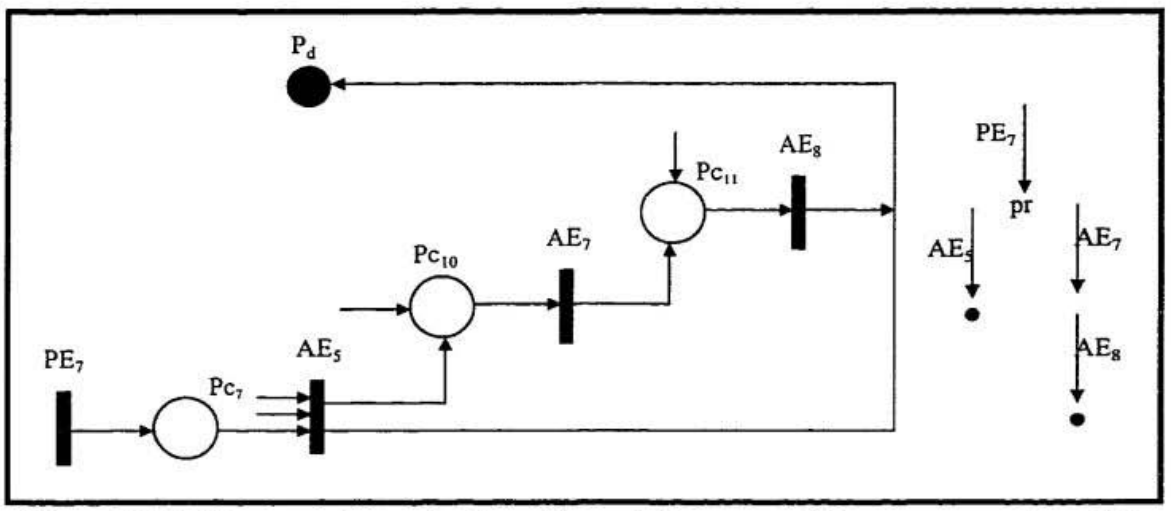


Figure B.10: The execution paths from the periodic process PE_7 to serve sensing of the seventh sensor.

The execution paths to serve the seventh sensor are defined by the following equations

$$S_7.p_1 = PE_7,\ AE_5 \qquad (B.43)$$

$$S_7.p_2 = PE_7,\ AE_7,\ AE_8 \qquad (B.44)$$

The maximum total time for periodic sensing of the seventh sensor is

$$S_7.t = PE_7.t + (AE_5.t + AE_7.t + AE_8.t) \qquad (B.45)$$

for sequential execution of parallelizable processes, or

$$S_7.t = PE_7.t + \max.\ \{AE_5.t,\ (AE_7.t + AE_8.t)\} \qquad (B.46)$$

for parallel execution of parallelizable processes.

B.3 Repetitiveness and Reachability Analysis

To facilitate the reachability and repetitiveness analysis the service sequence of different events are summarized in tabular form as shown in Table B.2.

Table B.2: The summary of the execution path analysis.

Execution paths	Periodic Events							Aperiodic Events							
	PE_1	PE_2	PE_3	PE_4	PE_5	PE_6	PE_7	AE_1	AE_2	AE_3	AE_4	AE_5	AE_6	AE_7	AE_8
$S_1.p_1$=PE_1,AE_1	x							x							
$S_1.p_2$=PE_1,AE_4,AE_7,AE_8	x										x			x	x
$S_1.p_3$=PE_1,AE_4,AE_6,AE_8	x										x		x		x
$S_2.p_1$=PE_2, AE_1		x						x							
$S_2.p_2$=PE_2, AE_4, AE_6,AE_8		x									x		x		x
$S_2.p_3$= PE_2, AE_4, AE_7, AE_8		x									x			x	x
$S_2.p_4$=PE_2, AE_2, AE_6, AE_8		x							x				x		x
$S_2.p_5$= PE_2, AE_3, AE_6, AE_8		x								x			x		x
$S_3.p_1$=PE_3, AE_4, AE_6, AE_8			x								x		x		x
$S_3.p_2$=PE_3, AE_4, AE_7, AE_8			x								x			x	x
$S_4.p_1$=PE_4, AE_2, AE_6, AE_8				x					x				x		x
$S_4.p_2$=PE_4, AE_5				x								x			
$S_4.p_3$=PE_4, AE_5, AE_7, AE_8				x								x		x	x
$S_5.p_1$=PE_5, AE_2, AE_6, AE_8					x				x				x		x
$S_5.p_2$=PE_5, AE_5, AE_7, AE_8					x							x		x	x
$S_5.p_3$=PE_5, AE_4, AE_6, AE_8					x						x		x		x
$S_5.p_4$=PE_5, AE_4, AE_7, AE_8					x						x			x	x
$S_6.p_1$=PE_6, AE_5						x						x			
$S_6.p_2$=PE_6, AE_7, AE_8						x								x	x
$S_7.p_1$=PE_7, AE_5							x					x			
$S_7.p_2$=PE_7, AE_7, AE_8							x							x	x

A few salient features of this formalism can be highlighted from the design data summarized in the above table. It should be noted that this proposed design formalism of SFS considers that for each sensor there must be a unique periodic event. The validity of the design to satisfy this proposition can be justified by checking the columns of the periodic events for each execution path. To satisfy this proposition each execution path serves only one periodic event (i.e., only one cell in the periodic events area is marked for each execution path).

In this proposed framework of SFS, the sensing operation is periodic in nature. To verify this repetitiveness criterion a simplified model of the SFS is shown in Fig.B.11. In this simplified representation only the terminal events (AE_1,AE_8, AE_5) and the periodic events are shown.

The periodic events (e.g., PE_1,...,PE_7) are generated at regular intervals of time. Now if there is at least one execution path from each periodic event ending in any one of the terminal events (e.g., AE_1,AE_8,AE_5), the operations of the designed SFS will be periodic in nature. To satisfy this requirement at least one cell of the column of each periodic event should be marked and at least one shaded cell of the same row should be marked. The design data summarized in the table for this example SFS satisfy this criterion. Therefore, this designed SFS is repetitive.

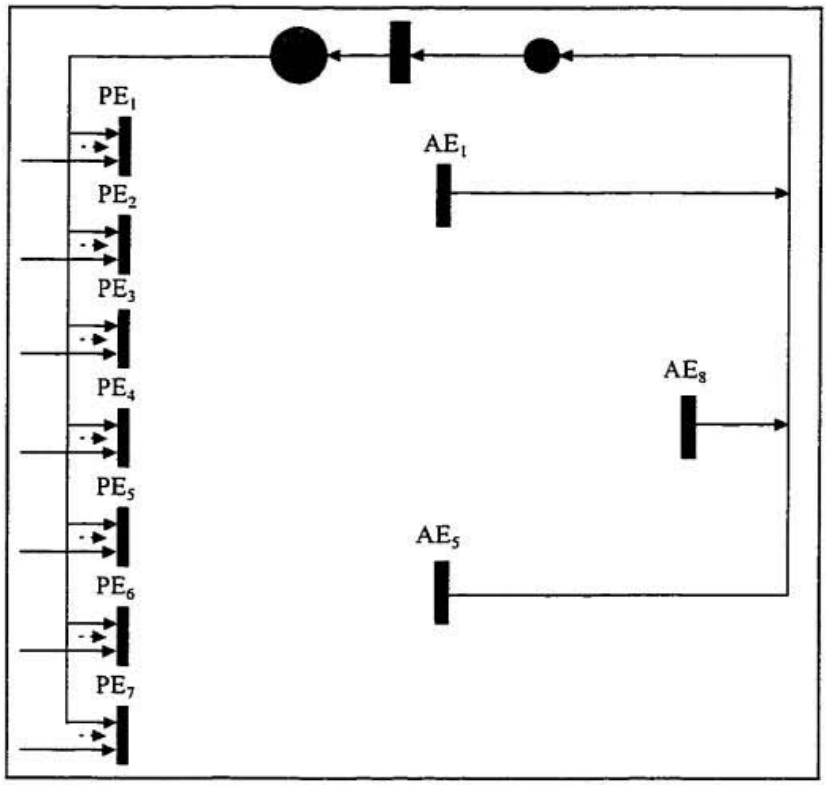


Figure B.11: The Simplified Petri net model of the example SFS to understand the problem of verification of repetitiveness

If each member of all the events (both periodic and aperiodic) is executed by at least one execution path, it can be concluded that every event is reachable. To satisfy this proposition each column of the events as shown in the table must be checked at least one and this is satisfied for the design of this example SFS.

B.4 The Sensing Sequence Analysis

The total sensing time of each sensor considering minimum event service time and sequential execution of the corresponding processes is shown in the following Table B.3.

Table B.3: The summary of sensing time estimation.

Sensors	Execution time relating equations	Sensing time	
S_1	$S_1.t = \max\{(PE_1.t+AE_1.t), (PE_1.t+AE_4.t+AE_7.t+AE_6.t+AE_8.t)\}$ $=\max.\{(10+12), (10+16+12+15)\}$	max..(22,53)	53
S_2	$S_2.t = PE_2.t+\max.\{(AE_1), (AE_4+AE_6+AE_7+AE_8)\}+$ $\max.\{(AE_2+AE_6+AE_8), (AE_3+AE_6+AE_8)$ $=18+\max.\{(12), (16+18+12+15)\}+\max.\{14,15\}+18+15$	18+max.(12, 61) +15+33	125
S_3	$S_3.t = PE_3.t+AE_4.t+AE_6+AE_7.t+AE_8.t$ $=12+16+18+15$	61	61
S_4	$S_4.t = PE_4.t+\max.\{(AE_2.t+AE_6.t+AE_8.t), AE_5.t, (AE_5.t+$ $AE_7.t+AE_8.t)\}$ $=14+\max.\{(14+18+15), 20, (20+12+15)\}$	14+ max. (47,47)	61
S_5	$S_5.t = PE_5.t+\max.\{(AE_2.t+AE_6.t), (AE_5.t+AE_7.t)\}+$ $AE_4.t+(AE_6.t+AE_7.t)+AE_8.t$ $=16+\max.\{(14+18), (20+12)\}+16+(18+12)+15$	16+max.(32, 32)+16+30+15	109
S_6	$S_6.t = PE_6.t+(AE_5.t+AE_7.t+AE_8.t)$ $=20+20+12+15$	67	67
S_7	$S_7.t = PE_7.t+(AE_5.t+AE_7.t+AE_8.t)$ $=25+20+12+15$	72	72

The time for house keeping operation is more or less constant for each sensing sequence and has been assumed to be zero for simplicity in remaining calculations. The distribution of the sensing time of each sensor can be shown in graphical form as in Fig.B.12. Here, the minimum sensing time is 53 units for S_1 and maximum sensing time is 125 units for S_2.

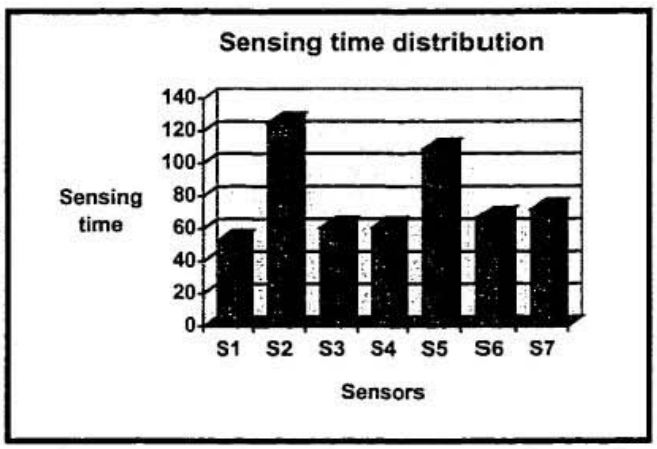


Figure B.12. Distribution of sensing times of different sensors.

A plot of the sensing sequence using initially user's defined sensing phase (max. phase), frequency (min. frequency) is shown in Fig.B.13. This plot shows the available and required total sensing time for each sensor.

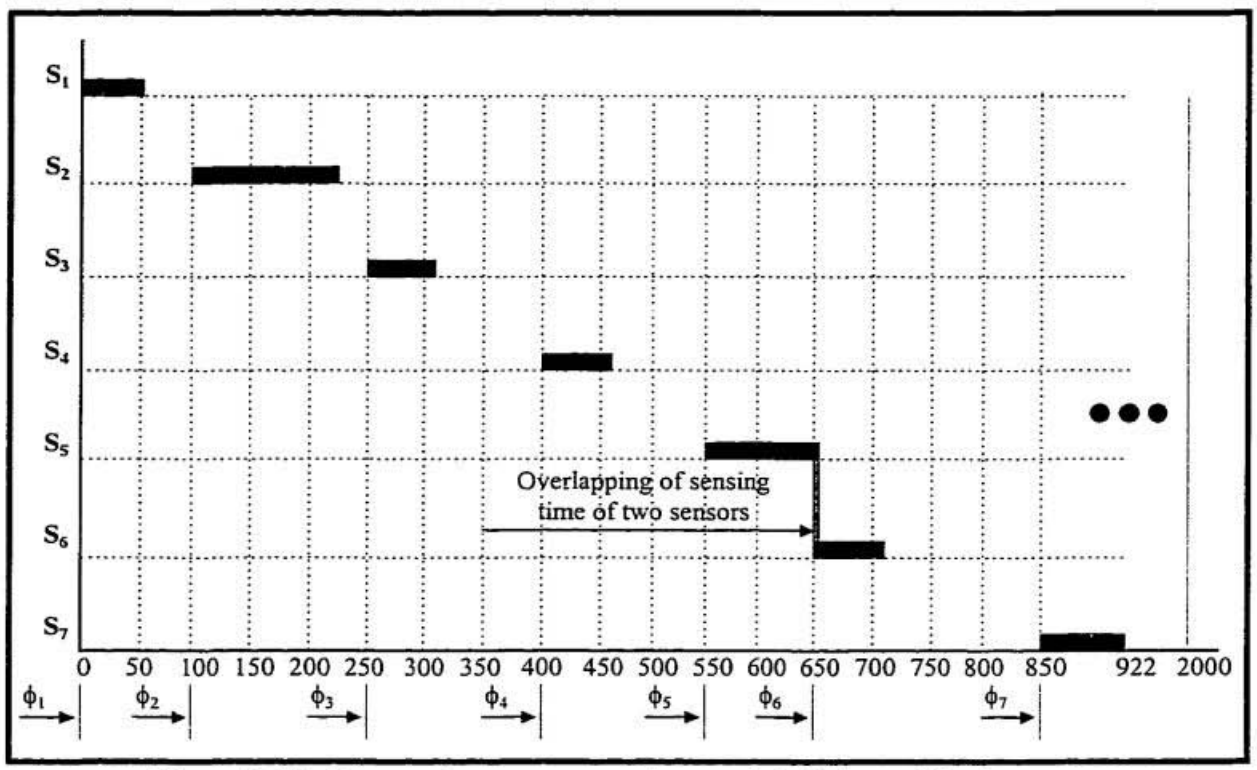


Figure B.13: The sensing sequence using user's initial specification.

From this plot it appears that in the first sequence of sensing, except one instance, there is no overlapping of sensing times and the first sequence of sensing finishes before the beginning of the second sequence. Therefore, the distribution of phase needs little modification to avoid this overlapping to make the SFS implementable on a single processor.

It is now worthwhile to go back to the proposed theory of sensing phase and periods distribution. From the initial investigation of the specified phase it seems that the phase grain size, gp=150 satisfies the initial condition, which is larger than the largest sensing time, 125. Now, all the phases should be integral multiple of this grain. Therefore, the proposed modified phases are shown in the Table B.4.

Table B.4: The modified phases of the sensors.

	Sensors						
	S_1	S_2	S_3	S_4	S_5	S_6	S_7
Phases	0	150	300	450	600	750	900

This modification in phase value will avoid detected overlapping of sensing time in the first sequence as shown in the previous figure. If the client agrees with this modification, the first sequence of sensing is free from overlapping.

Now according to the theory, the grain size of the periods should be larger than the sum of the longest phase and the corresponding sensing time, and should be an integral multiple of phase grain size. In this case, Gp=1050 is a reasonable choice. Now all the periods should be integral multiple of this grain size. To satisfy this condition the slightly modified periods are shown in the following Table B.5.

Table B.5: The periods of the sensors.

	Sensors						
	Ts_1	Ts_2	Ts_3	Ts_4	Ts_5	Ts_6	Ts_7
Periods	2100	2100	3150	3150	3150	3150	1050

If the client agrees with this modification, according to the theory there will be no overlap in the sensing time. This can be verified by plotting the sensing time in the second sequence of sensing as shown in Fig. B.14.

With respect to this design data this SFS can accommodate a maximum of 7 sensors. The sensor's number can be increased either by decreasing the maximum sensing time (in this case the sensing time of the second sensor is 125) or increasing the smallest period. It should be noted that in the 2nd sequence of sensing the system is idle for a long period of time. The CPU utilization factor of this system is shown below

$$U(3900+67)=\frac{(53+125+61+61+109+67+72)+(72+53+125+61+61+109+67)}{3900+67}=\frac{1096}{3967}=27.67\% \quad \text{(B.47)}$$

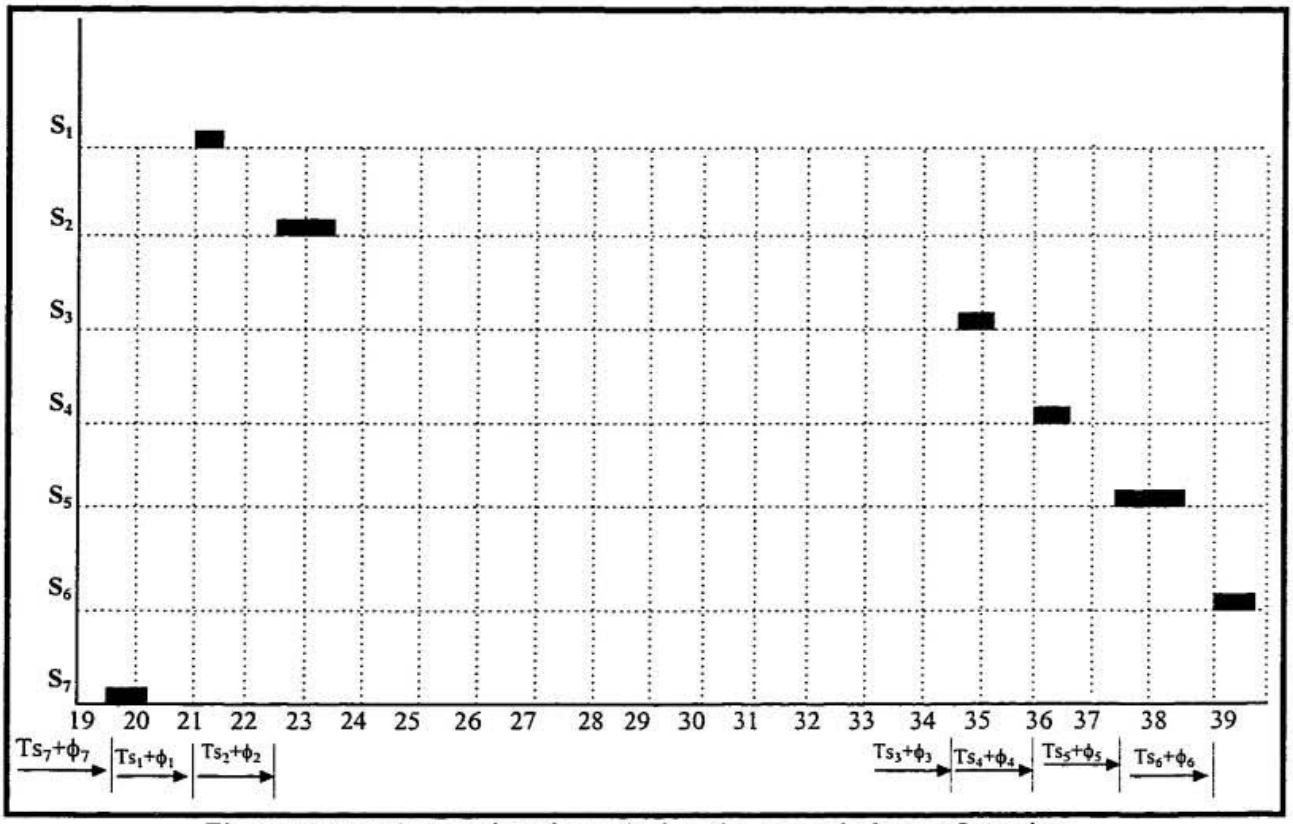


Figure B.14: The sensing times during the second phase of sensing.

This simulation study has shown the use of the proposed novel technique (chapter 2) to model the requirements of the example sensor fusion system (Appendix A). Through the execution path analysis technique the temporal and logical correctness of the modeled SFS has been studied. The repetitiveness and reachability analysis has been performed. The overlapping of the sensing periods has been identified by the analysis of sensing sequence and that has been avoided by the use of the technique proposed in chapter 2. The resource utilization factor of this example SFS has been measured by the proposed method (section 2.7).

Appendix C

Verification of Discrete Event Specifications Model of SFS by Simulation

C.1 Introduction

Every aperiodic event of the DEVR model as shown in appendix B will be served by the interactions of a set of computing components. It has been assumed that the size of this set is *16*. The lower and upper limits of computation times of these components are shown in Table C.1. These limits have been selected randomly and vary from *1* to *4* units of time. The input and output conditions of these components are selected randomly as well and shown in the same table. The sequential, branching, looping and parallel operations of the computing components to serve an event have been considered here. It has been assumed that every event in DEVR model has appropriate unique I/O interfaces requiring very small execution times.

Table C.1: The specification of a set of computing components.

Components	Execution times		I/O conditions	
	Lower limit	Upper limit	Inputs	Outputs
c_1	3.6	3.6	2	1
c_2	3.9	4.0	3	2
c_3	3.8	4.0	1	2
c_4	3.2	3.4	3	1
c_5	4.0	4.0	1	1
c_6	2.8	3.0	2	1
c_7	1.8	2.0	2	2
c_8	3.2	3.6	2	3
c_9	3.4	3.8	1	2
c_{10}	2.1	2.3	2	1
c_{11}	3.2	3.4	2	2
c_{12}	3.2	4.0	2	1
c_{13}	2.0	2.1	1	1
c_{14}	2.7	3.0	1	2
c_{15}	3.5	3.7	2	1
c_{16}	2.8	3.0	1	1

C.2 The Decomposition of Aperiodic Events in Terms of Interactions among the Computing Components

The eight aperiodic events of the DEVR model of the example sensor fusion system as modeled in appendix B will be served by the execution of a set of computing components as shown in Table C.1. Each execution path of these interactions is analyzed to compute the total execution times required to serve these aperiodic events.

C.2.1 The Decomposition of the Aperiodic Event AE_1

The representation of the aperiodic event AE_1 with the interaction of the computing components is shown in Fig. C.1.

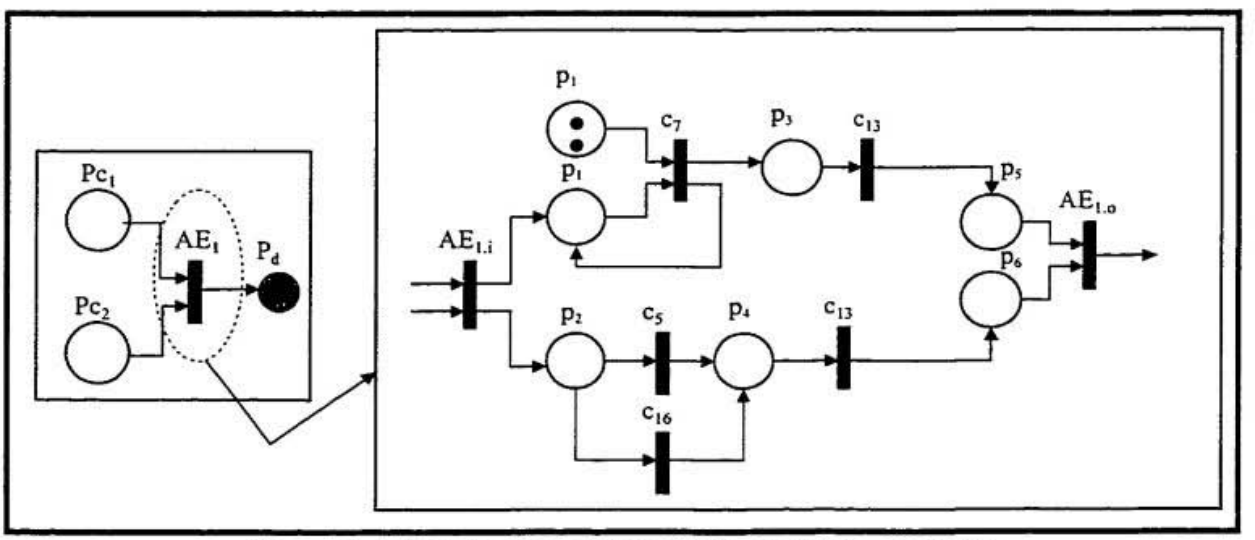


Figure C.1: The decomposition of the aperiodic event AE_1.

The execution paths to serve the aperiodic event AE_1 is shown by the following equations:

$$S_{AE_1}.p_1=AE_{1.i}, 2c_7, c_{13}, AE_{1.o} \quad (C.1)$$

$$S_{AE_1}.p_2=AE_{1.i}, c_5, c_{13}, AE_{1.o} \quad (C.2)$$

$$S_{AE_1}.p_2=AE_{1.i}, c_{16}, c_{13}, AE_{1.o} \quad (C.3)$$

The maximum total computation time to serve the event AE_1 is

$$S_{AE_1}.t = AE_{1.i}.t + 2c_7.t + c_{13}.t + \max.(c_5.t, c_{16}.t) + c_{13}.t + AE_{1.o}.t \quad (C.4)$$

for sequential execution of parallelizable components, or

$$S_{AE_1}.t = AE_{1.i}.t + \max.[\{2c_7.t + c_{13}.t\}, \{\max.(c_5.t, c_{16}.t) + c_{13}.t\}] + AE_{1.o}.t \quad (C.5)$$

for parallel execution of parallelizable components.

The objective is to maximize the execution time of each computing component with the constraint that the total execution time of the constituting components does not exceed the allocated service time of the corresponding aperiodic event in the DEVR model. The maximum and minimum values of total computation time to serve the event AE_1 for sequential execution of parallelizable components are shown by the following two relations.

$$S_{AE_1}.t_{max} = 0 + 2x2.0 + 2.1 + \max.(4.0, 3.0) + 2.1 = 4.0 + 2.1 + 4.0 + 2.1 = 12.2 \quad (C.6)$$

$$S_{AE_1}.t_{min} = 0 + 2x1.8 + 2.0 + \max.(4.0, 2.8) + 2.0 = 3.6 + 2.0 + 4.0 + 2.0 = 11.6 \quad (C.7)$$

From the above two equations, it appears that $S_{AE_1}.t_{max}$ exceeds the allocated service time of AE_1 specified by the DEVR model, but $S_{AE_1}.t_{min}$ is lower than the allocated total time. Here, a decision should be made to reduce the computation times of appropriate computing components. It should be noted that these components will be used to serve other aperiodic events within temporal constraints defined by the DEVR model. A few salient features of the effect of the reduction of computing time of a component on the total execution time can be explained by an example. For example, if the execution time of c_7 is reduced by .1 unit, the total execution time of the components to serve the event AE_1 is reduced by .1x2=.2 units. On the other hand, the reduction of execution time of the component c_{16} from 3.0 units to 2.8 units does not have any effect on the total execution time. The effect of the reduction of computing time of each computing component on the service times of all aperiodic events should be studied before the selection of computing time of any component to maximize the total reduction of execution times to serve all the aperiodic events.

C.2.2 The Decomposition of the Aperiodic Event AE_2

The representation of the aperiodic event AE_2 with the interaction of the computing components is shown in Fig. C.2.

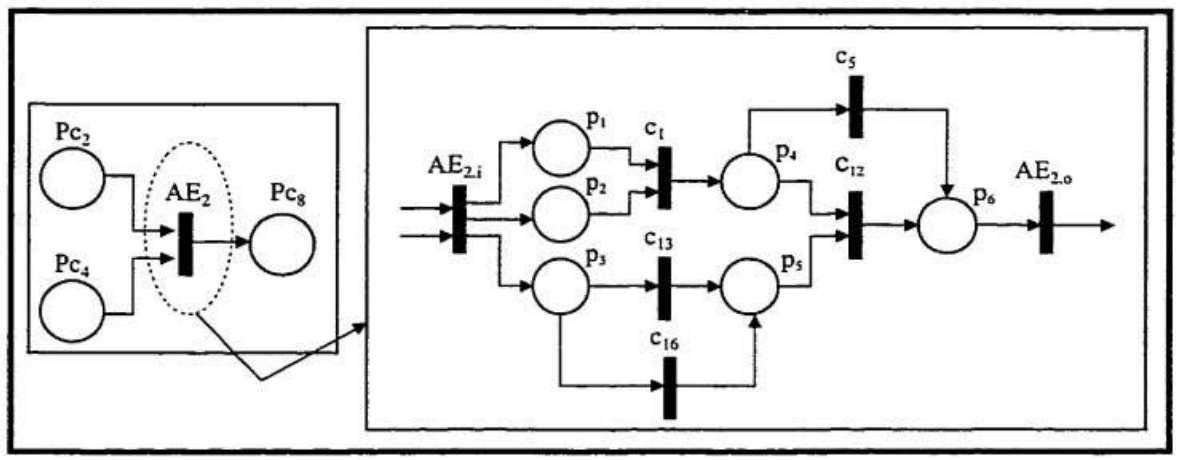


Figure C.2: The decomposition of the aperiodic event AE_2.

The execution paths to serve the aperiodic event AE_2 is shown by the following equations:

$$S_{AE_2.p_1}=AE_{2.i},\ c_1,c_5,AE_{2.o} \tag{C.8}$$

$$S_{AE_2.p_2}=AE_{2.i},\ c_1,c_{12},\ AE_{2.o} \tag{C.9}$$

$$S_{AE_2.p_3}=AE_{2.i},\ c_{13},c_{12},\ AE_{2.o} \tag{C.10}$$

$$S_{AE_2.p_4}=AE_{2.i},\ c_{16},c_{12},\ AE_{2.o} \tag{C.11}$$

The maximum total computation time to serve the event AE_2 is

$$S_{AE_2}.t= AE_{2.i}.t+ c_1.t+c_5.t+\max.(c_{13}.t,\ c_{16}.t)+c_{12}.t +AE_{2.o}.t \tag{C.12}$$

for sequential execution of parallelizable components, or

$$S_{AE_2}.t= AE_{2.i}.t+ \max.\ \{c_1.t,\ \max.(c_{13}.t,\ c_{16}.t)\}+\max.\ (c_5.t,\ c_{12}.t\} +AE_{2.o}.t \tag{C.13}$$

for parallel execution of parallelizable components.

Therefore, using the sequential execution of all the components

$$S_{AE2}.t_{max}=0+3.6+4.0+\max.(2.1, 3.0)+4.0+0=7.6+3.0+4.0=14.6 \quad (C.14)$$

$$S_{AE2}.t_{min}=0+3.6+4.0+\max.(2.0, 2.8)+3.2+0=7.6+2.8+3.2=13.6 \quad (C.15)$$

The $S_{AE2}.t_{max}$ exceeds the allocated lower limit of service time for AE2, but it is smaller than the upper limit of the allocated time. On the other hand, $S_{AE2}.t_{min}$ is even lower than the lower limit of the allocated service time.

C.2.3 The Decomposition of the Aperiodic Event AE_3

The representation of the aperiodic event AE_3 with the interactions of the computing components is shown in Fig. C.3. The execution paths to serve the event AE_3 is shown by the following equations:

$$S_{AE_3.p_1}=AE_3.i, c_{15}, c_5, AE_{3.o} \quad (C.16)$$

$$S_{AE_3.p_2}=AE_3.i, c_{15}, c_{12}, AE_{3.o} \quad (C.17)$$

$$S_{AE_3.p_3}=AE_3.i, c_{13}, c_{12}, AE_{3.o} \quad (C.18)$$

$$S_{AE_3.p_4}=AE_3.i, c_{11}, c_{12}, AE_{3.o} \quad (C.19)$$

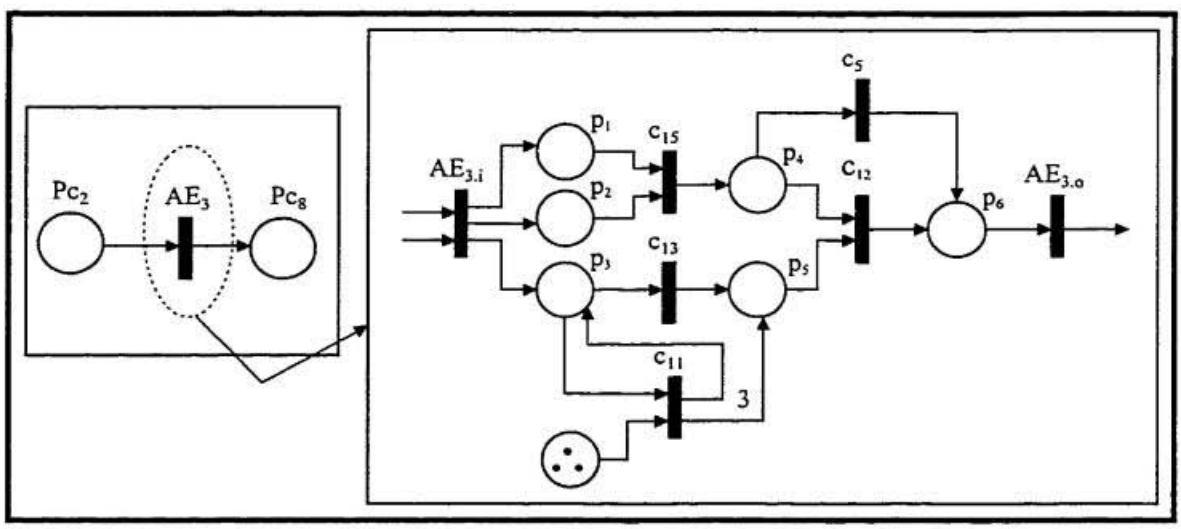


Figure C.3: The decomposition of the aperiodic event AE_3.

The maximum total computation time to serve the event AE_3 is

$$S_{AE_3}.t = AE_{3.i}.t + c_{15}.t + c_5.t + c_{12}.t + \max.(c_{13}.t, 3c_{11}.t) + AE_{3.o}.t \quad \text{(C.20)}$$

for sequential execution of parallelizable components, or

$$S_{AE_3}.t = AE_{3.i}.t + \max.\{c_{15}.t, \max.(c_{13}.t, 3c_{11}.t)\} + \max.(c_5.t, c_{12}.t) + AE_{3.o}.t \quad \text{(C.21)}$$

for parallel execution of parallelizable components.

Therefore, using the sequential execution of all the components

$$S_{AE_3}.t_{min} = 0+3.5+4.0+3.2+\max.(2.0, 3x3.2)+0 = 10.7+9.6 = 20.3, \text{ and} \quad \text{(C.22)}$$

$$S_{AE_3}.t_{max} = 0+3.7+4.0+4.0+\max.(2.1, 3x3.4)+0 = 11.7+10.2 = 21.9 \quad \text{(C.23)}$$

Here, the minimum execution time (20.3 units) to serve the aperiodic event AE_3 is higher than the upper limit (19 units) of the service time defined in the DEVR model.

C.2.4 The Decomposition of the Aperiodic Event AE_4

The representation of the aperiodic event AE_4 with the interaction of the computing components is shown in Fig. C.4.

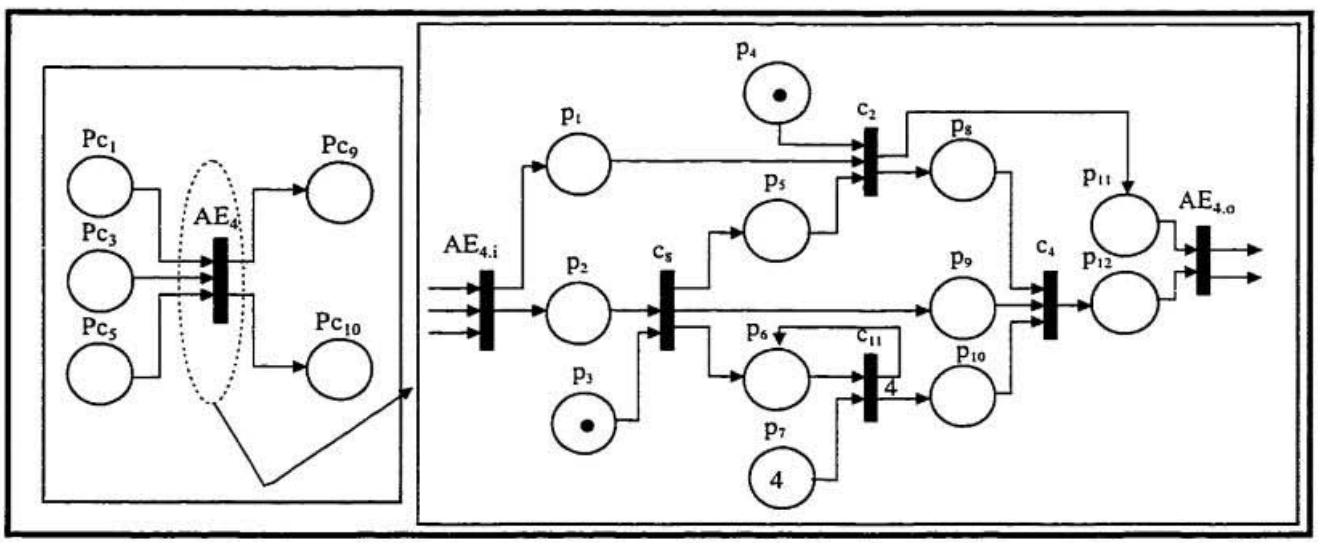


Figure C.4: The decomposition of the aperiodic event AE_4.

The execution paths to serve the aperiodic event AE_4 is shown by the following equations:

$$S_{AE_4}.p_1 = AE_{4.i}, c_2, AE_{4.o} \quad \text{(C.24)}$$

$$S_{AE_4}.p_2 = AE_{4.i}, c_8, c_2, AE_{4.o} \quad \text{(C.25)}$$

$$S_{AE_4}.p_3 = AE_{4.i}, c_8, c_4, AE_{4.o} \quad \text{(C.26)}$$

$$S_{AE_4}.p_4 = AE_{4.i}, c_8, c_{11}, c_4, AE_{4.o} \quad \text{(C.27)}$$

The maximum total computation time to serve the event AE4 is

$$S_{AE_4}.t = AE_{4.i}.t + c_8.t + c_2.t + 4\,c_{11}.t + c_4.t + AE_{4.o}.t \quad \text{(C.28)}$$

for sequential execution of parallelizable components, or

$$S_{AE_4}.t = AE_{4.i}.t + c_8.t + \max.(c_2.t, 4\,c_{11}.t) + c_4.t + AE_{4.o}.t \quad \text{(C.29)}$$

for parallel execution of parallelizable components.

Therefore, using the sequential execution of all the components

$$S_{AE_4}.t_{max} = 0.0+3.6+4.0+4x3.4+3.4+0.0 = 7.6+13.6+3.4 = 24.6 \quad \text{(C.30)}$$

$$S_{AE_4}.t_{min} = 0.0+3.2+3.9+4x\ 3.2+3.2+0.0 = 7.1+12.8+3.2 = 23.1 \quad \text{(C.31)}$$

C.2.5 The Decomposition of the Aperiodic Event AE_5

The representation of the aperiodic event AE_5 with the interaction of the computing components is shown in Fig. C.5.

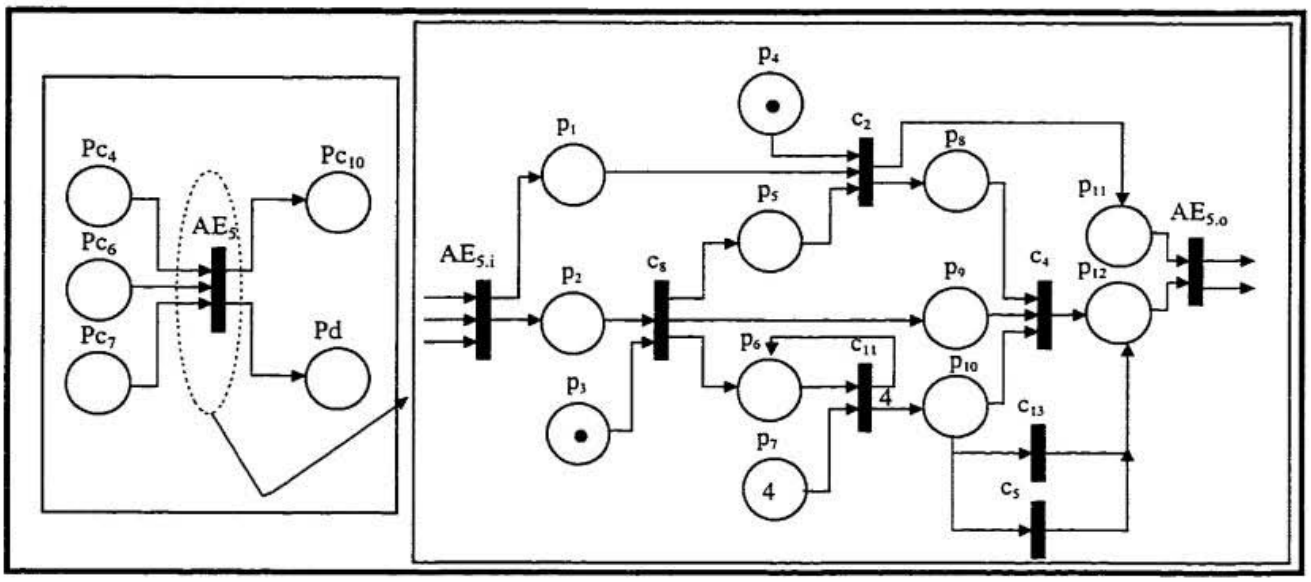


Figure C.5: The decomposition of the aperiodic event AE_5.

The execution paths to serve the aperiodic event AE_5 is shown by the following equations:

$$S_{AE_5}.p_1=AE_{5.i}, c_2 , AE_{5.o} \quad (C.32)$$

$$S_{AE_5}.p_2=AE_{5.i}, c_8 , c_2 , AE_{5.o} \quad (C.33)$$

$$S_{AE_5}.p_3=AE_{5.i}, c_8 , c_4 , AE_{5.o} \quad (C.34)$$

$$S_{AE_5}.p_4=AE_{5.i}, c_8 , c_{11} , c_4, AE_{5.o} \quad (C.35)$$

$$S_{AE_5}.p_5=AE_{5.i}, c_8 , c_{11} , c_{13}, AE_{5.o} \quad (C.36)$$

$$S_{AE_5}.p_6=AE_{5.i}, c_8 , c_{11} , c_5, AE_{5.o} \quad (C.37)$$

The maximum total computation time to serve the event AE_5 is

$$S_{AE_5}.t = AE_{5.i}.t + c_8 .t + c_2.t + 4\, c_{11}.t + \max.(c_4.t, c_{13}.t, c_5.t) + AE_{5.o}.t \quad (C.38)$$

for sequential execution of parallelizable components, or

$$S_{AE_5}.t = AE_{5.i}.t + c_8 .t + \max.(c_2.t , 4\, c_{11}.t) + \max.(c_4.t, c_{13}.t, c_5.t) + AE_{5.o}.t \quad (C.39)$$

for parallel execution of parallelizable components.

Therefore, using the sequential execution of all the components

$$S_{AE_5}.t_{max} = 0.0+3.6+4.0+4x3.4+\max.(3.4, 2.1, 4.0)+0.0 = 7.6+13.6+4.0 = 25.2 \quad (C.40)$$

$$S_{AE_5}.t_{min} = 0.0+3.2+3.9+4x\,3.2+\max.(3.2, 2.0, 4.0)+0.0 = 7.1+12.8+4.0 = 23.9 \quad (C.41)$$

C.2.6 The Decomposition of the Aperiodic Event AE_6

The representation of the aperiodic event AE_6 with the interaction of the computing components is shown in Fig. C.6.

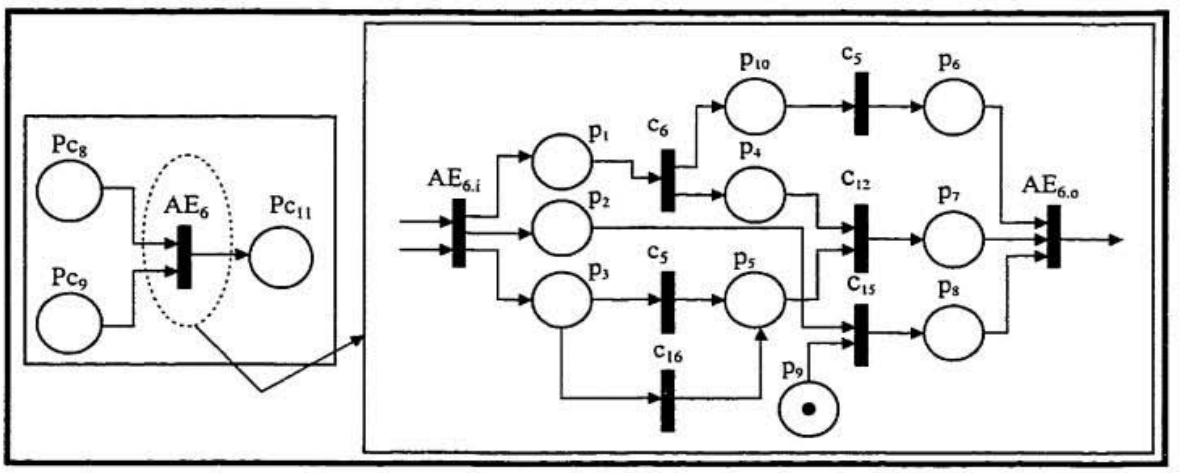


Figure C.6: The decomposition of the aperiodic event AE_6.

The execution paths to serve the aperiodic event AE_6 is shown by the following equations:

$$S_{AE_6}.p_1 = AE_{6.i}, c_6, c_5, AE_{6.o} \quad \text{(C.42)}$$

$$S_{AE_6}.p_2 = AE_{6.i}, c_6, c_{12}, AE_{6.o} \quad \text{(C.43)}$$

$$S_{AE_6}.p_3 = AE_{6.i}, c_{15}, AE_{6.o} \quad \text{(C.44)}$$

$$S_{AE_6}.p_4 = AE_{6.i}, c_5, c_{12}, AE_{6.o} \quad \text{(C.45)}$$

$$S_{AE_6}.p_5 = AE_{6.i}, c_{16}, c_{12}, AE_{6.o} \quad \text{(C.46)}$$

The maximum total computation time to serve the event AE_6 is

$$S_{AE_6}.t = AE_{6.i}.t + c_6.t + \max.(c_5.t, c_{16}.t) + c_5.t + c_{12}.t + c_{15}.t + AE_{6.o}.t \quad \text{(C.47)}$$

for sequential execution of parallelizable components, or

$$S_{AE_6}.t = AE_{6.i}.t + \max.\{ c_6.t, \max.(c_5.t, c_{16}.t) \} + \max.(c_5.t, c_{12}.t, c_{15}.t) + AE_{6.o}.t \quad \text{(C.48)}$$

for parallel execution of parallelizable components.

Therefore, using the sequential execution of all the components

$$S_{AE_6}.t_{max} = 0.0+3.0+\max.(4.0, 3.0)+4.0+4.0+3.7+0.0 = 3.0+4.0+11.7 = 18.7 \quad \text{(C.49)}$$

$$S_{AE_6}.t_{min} = 0.0+2.8+\max.(4.0, 2.8)+4.0+3.2+3.5+0.0 = 2.8+4.0+10.7 = 17.5 \quad \text{(C.50)}$$

C.2.7 The Decomposition of the Aperiodic Event AE_7

The representation of the aperiodic event AE_7 with the interaction of the computing components is shown in Fig. C.7.

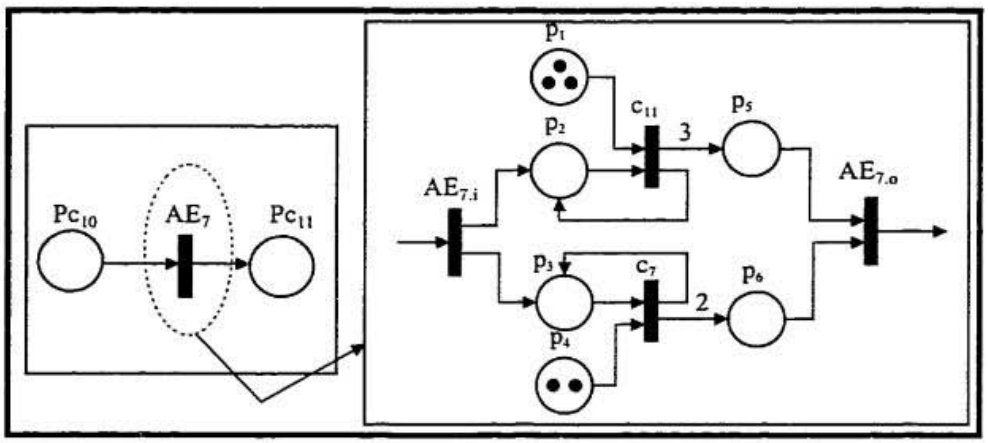


Figure C.7: The decomposition of the aperiodic event AE_7.

The execution paths to serve the aperiodic event AE_7 is shown by the following equations:

$$S_{AE_7}.p_1 = AE_{7.i}, c_{11}, AE_{7.o} \quad \text{(C.51)}$$

$$S_{AE_7}.p_2 = AE_{7.i}, c_7, AE_{7.o} \quad \text{(C.52)}$$

The maximum total computation time to serve the event AE_7 is

$$S_{AE_7}.t = AE_{7.i}.t + 3\ c_{11}\ .t + 2\ c_7.t + AE_{7.o}.t \quad \text{(C.53)}$$

for sequential execution of parallelizable components, or

$$S_{AE_7}.t = AE_{7.i}.t + \max.\ (3\ c_{11}\ .t\ ,\ 2\ c_7.t\ ) + AE_{7.o}.t \quad \text{(C.54)}$$

for parallel execution of parallelizable components.

Therefore, using the sequential execution of all the components

$$S_{AE_7}.t_{max} = 0.0 + 3x3.4 + 2x2.0 + 0.0 = 10.2 + 4.0 = 14.2 \quad \text{(C.55)}$$

$$S_{AE_7}.t_{min} = 0.0 + 3x3.2 + 2x1.8 + 0.0 = 9.6 + 3.6 = 13.2 \quad \text{(C.56)}$$

C.2.8 The Decomposition of the Aperiodic Event AE_8

The representation of the aperiodic event AE_8 with the interaction of the computing components is shown in Fig. C.8.

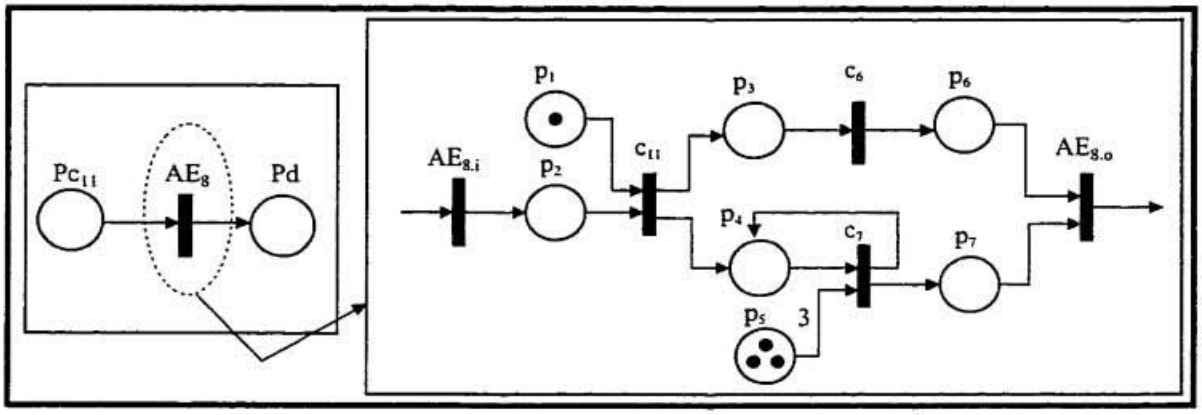


Figure C.8: The decomposition of the aperiodic event AE_8.

The execution paths to serve the aperiodic event AE_8 is shown by the following equations:

$$S_{AE_8}.p_1 = AE_{8.i},\ c_{11},\ c_6\ AE_{8.o} \tag{C.57}$$

$$S_{AE_8}.p_2 = AE_{7.i},\ c_{11},\ c_7,\ AE_{8.o} \tag{C.58}$$

The maximum total computation time to serve the event AE_8 is

$$S_{AE_8}.t = AE_{8.i}.t + c_{11}\ .t + c_6.t + 3\ c_7.t + AE_{8.o}.t \tag{C.59}$$

for sequential execution of parallelizable components, or

$$S_{AE_8}.t = AE_{8.i}.t + c_{11}.t + \max.\ (c_6.t,\ 3\ c_7\ .t) + AE_{8.o}.t \tag{C.60}$$

for parallel execution of parallelizable components.

Therefore, using the sequential execution of all the components

$$S_{AE_8}.t_{max} = 0.0+3.4+3.0+3x2.0+0.0 = 3.4+3+6.0 = 12.4 \tag{C.61}$$

$$S_{AE_8}.t_{min} = 0.0+3.2+2.8+3x1.8 = 6.0+5.4 = 11.4 \tag{C.62}$$

C.3 The Optimization of the Execution Times of the Computing Components

For this optimization problem, the minimum service times for the aperiodic processes have been considered. The specified service times for the aperiodic events, attainable service times considering minimum and maximum executions of the corresponding computing components are shown in Table C.2. For this optimization problem, it has been assumed that the parallellizable components have been executed sequentially on a single processor based computing system.

Table C.2: The minimum service times of the aperiodic events and the corresponding attainable service times.

Aperiodic processes	Allocated service time	Attainable minimum execution times	Attainable maximum execution times
AE_1	12	11.6	12.2
AE_2	14	13.6	14.6
AE_3	15	20.3	21.9
AE_4	16	23.1	24.6
AE_5	20	23.9	25.2
AE_6	18	17.5	18.7
AE_7	12	13.2	14.2
AE_8	15	11.4	12.4

The allocated service times for AE_1, AE_2 and AE_6 are within the range of attainable minimum and maximum execution times. For these aperiodic processes, the optimization technique as proposed in section 3.3 will maximize the allocated execution times of the corresponding computing components. For AE_8, the allocated service time is even greater than possible maximum execution times of the corresponding components. Therefore, optimization technique will have no effect on allocation time of this event.

The allocated service times of AE_3, AE_4, AE_5, and AE_7 are even higher than the attainable minimum execution times. In this case, it is the job of the designer to bring the allocated service times of these events within the range of the attainable minimum and maximum execution times. The designer can change the sensing strategy or redefine the interaction of components to serve

these events, or can negotiate with the clients to redefine the system requirements. For this present analysis, it has been assumed that due to the change of the system requirements the allocated service times of the aperiodic events AE_3, AE_4, AE_5, and AE_7 are 21, 24, 24 and 14 units respectively. The modified service times are shown in Table C.3.

Table C.3: The modified minimum service times of the aperiodic events and the corresponding attainable service times.

Aperiodic processes	Allocated service time	Attainable minimum execution times	Attainable maximum execution times
AE_1	12	11.6	12.2
AE_2	14	13.6	14.6
AE_3	21	20.3	21.9
AE_4	24	23.1	24.6
AE_5	24	23.9	25.2
AE_6	18	17.5	18.7
AE_7	14	13.2	14.2
AE_8	15	11.4	12.4

The equations to calculate the total execution times of the corresponding computing components to serve these aperiodic events are shown in the Table C.4.

Table C.4: The aperiodic events and the execution times of their corresponding computing traces to serve them.

Aperiodic events	The execution times of the traces to serve aperiodic events
AE_1	$S_{AE_1}.t = AE_{1.i}.t + 2c_7.t + c_{13}.t + \max._b(c_5.t, c_{16}.t) + c_{13}.t + AE_{1.o}.t$
AE_2	$S_{AE_2}.t = AE_{2.i}.t + c_1.t + c_5.t + \max._b(c_{13}.t, c_{16}.t) + c_{12}.t + AE_{2.0}.t$
AE_3	$S_{AE_3}.t = AE_{3.i}.t + c_{15}.t + c_5.t + c_{12}.t + \max._b(c_{13}.t, 3c_{11}.t) + AE_{3.o}.t$
AE_4	$S_{AE_4}.t = AE_{4.i}.t + c_8.t + c_2.t + 4\,c_{11}.t + c_4.t + AE_{4.o}.t$
AE_5	$S_{AE_5}.t = AE_{5.i}.t + c_8.t + c_2.t + 4\,c_{11}.t + \max._b(c_4.t, c_{13}.t, c_5.t) + AE_{5.o}.t$
AE_6	$S_{AE_6}.t = AE_{6.i}.t + c_6.t + \max._b(c_5.t, c_{16}.t) + c_5.t + c_{12}.t + c_{15}.t + AE_{6.o}.t$
AE_7	$S_{AE_7}.t = AE_{7.i}.t + 3\,c_{11}.t + 2\,c_7.t + AE_{7.o}.t$
AE_8	$S_{AE_8}.t = AE_{8.i}.t + c_{11}.t + c_6.t + 3\,c_7.t + AE_{8.o}.t$

Now the execution times of the corresponding computing components should be optimized according to the proposed technique as explained in Section 3.3.

From preliminary investigation, the components of the specified set as shown in Table C.1 which do not take part in serving these events can be sorted out as shown in Table C.5.

Table C.5: The aperiodic events and the corresponding computing components.

Events	Computing components															
	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}	c_{12}	c_{13}	c_{14}	c_{15}	c_{16}
AE_1					x		x						x			x
AE_2	x				x							x	x			x
AE_3					x						x	x	x		x	
AE_4		x		x				x			x					
AE_5		x		x	x			x			x		x			
AE_6					x	x						x			x	x
AE_7							x				x					
AE_8						x	x				x					

The components c_3, c_9, c_{10}, and c_{14} did not take part in serving these eight aperiodic events. Therefore, the optimization of their values will not be considered here.

It should be noted that there is max_b (maximum time of the branches) operator in the equations to compute the attainable service times of the aperiodic events by executing the corresponding components as shown in Table C.4. Therefore, the linear optimization will not work here. The flow chart of the adopted optimization algorithm is shown in Fig. C.9.

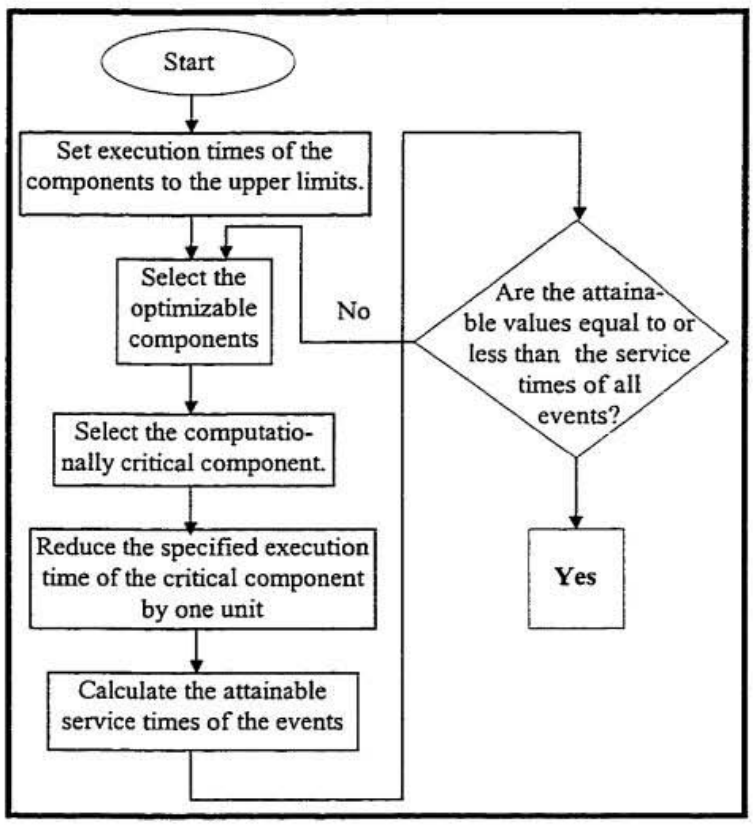


Figure C.9: Flow chart for optimization.

It took *21* iterations for the optimization of this present problem. After the optimization, the specified execution times of the computing components are shown in Table C.6.

Table C.6: Optimized execution times of the computing components.

Components	Execution times		Optimized times
	Lower limit	Upper limit	
c_1	3.6	3.6	3.6
c_2	3.9	4.0	3.9
c_3	3.8	4.0	4.0
c_4	3.2	3.4	3.4
c_5	4.0	4.0	4.0
c_6	2.8	3.0	2.8
c_7	1.8	2.0	1.8
c_8	3.2	3.6	3.3
c_9	3.4	3.8	3.8
c_{10}	2.1	2.3	2.3
c_{11}	3.2	3.4	3.2
c_{12}	3.2	4.0	3.2
c_{13}	2.0	2.1	2.0
c_{14}	2.7	3.0	3.0
c_{15}	3.5	3.7	3.5
c_{16}	2.8	3.0	3.0

The service times of the aperiodic events with these new optimized execution times of the computing components are shown in Table C.7.

Table C.7: The service times of the aperiodic events after optimization

Aperiodic processes	Allocated service time	Attainable service times after optimization
AE_1	12	11.6
AE_2	14	13.8
AE_3	21	20.30
AE_4	24	23.40
AE_5	24	24.00
AE_6	18	17.5
AE_7	14	13.2
AE_8	15	11.4

At each iteration, this optimization algorithm selects the temporally most critical component to maximize the overall reduction of the service times of all aperiodic events. The profile of the selection of these temporally critical components for this present optimization problem is shown in Table C.8.

Table C.8: The selection of temporally critical component at different interations.

	No of iterations																				
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
Components	c_{11}	c_{11}	c_7	c_7	c_{12}	c_{12}	c_{12}	c_{12}	c_{12}	c_{12}	c_{12}	c_{12}	c_2	c_8	c_8	c_{15}	c_{13}	c_{15}	c_6	c_6	c_8

The ratio of the total reduction of the service times of all the aperiodic components at each iteration to the decrement of execution time of the temporally critical component is shown in Fig.C.10.

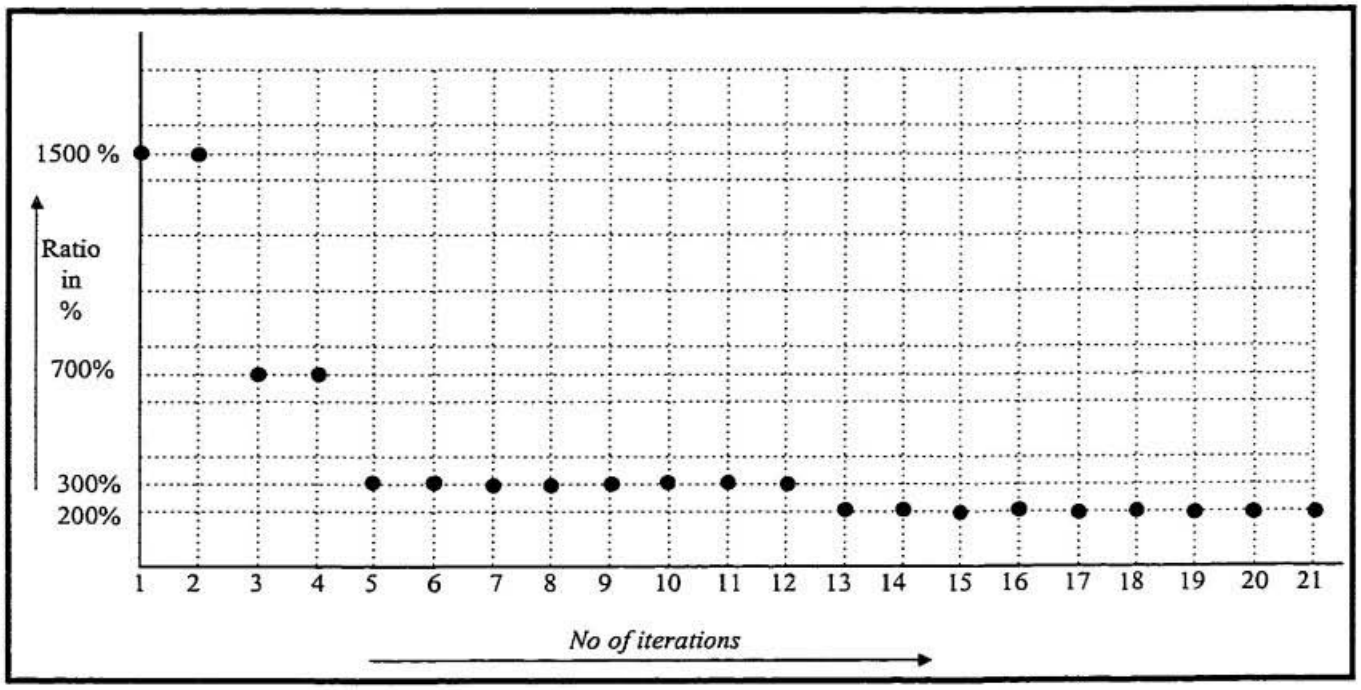


Figure C.10: The ratio of the total reduction of the service times of all aperiodic events to the decrement of the execution time of the temporally critical component.

This simulation study has shown the mapping of the DEVR model of the example SFS to components level specifications in an optimized way.

Appendix D

The Architecture of the Embedded Computing System to Implement the Example SFS

D.1 Introduction

The interactions of the computing components for the service of the aperiodic events to realize the example sensor fusion system as defined in the Appendix A have been shown in Appendix C. The decomposition of the execution paths as shown in sections C.2.1,...,C.2.8 reveals that the components interact in both sequential and parallel ways. It should be noted that parallelizable components can be executed both in sequential and parallel fashion. Depending upon the mode of execution of parallelizable components the architecture of the underlying computing system can be based on a single node or multiple nodes (processors).

D.2 The Architecture of the Computing System while Parallelizable Components are Executed in Sequential Fashion

Table D.1: The maximum total computation times to serve the aperiodic events for the sequential execution of parallelizable components.

Aperiodic events	The maximum computation times
$\mathbf{AE_1}$	$S_{AE_1}.t = AE_{1.i}.t + 2c_7.t + c_{13}.t + \max._b(c_5.t, c_{16}.t) + c_{13}.t + AE_{1.o}.t$
$\mathbf{AE_2}$	$S_{AE_2}.t = AE_{2.i}.t + c_1.t + c_5.t + \max._b(c_{13}.t, c_{16}.t) + c_{12}.t + AE_{2.0}.t$
$\mathbf{AE_3}$	$S_{AE_3}.t = AE_{3.i}.t + c_{15}.t + c_5.t + c_{12}.t + \max._b(c_{13}.t, 3c_{11}.t) + AE_{3.o}.t$
$\mathbf{AE_4}$	$S_{AE_4}.t = AE_{4.i}.t + c_8.t + c_2.t + 4\,c_{11}.t + c_4.t + AE_{4.o}.t$
$\mathbf{AE_5}$	$S_{AE_5}.t = AE_{5.i}.t + c_8.t + c_2.t + 4\,c_{11}.t + \max._b(c_4.t, c_{13}.t, c_5.t) + AE_{5.o}.t$
$\mathbf{AE_6}$	$S_{AE_6}.t = AE_{6.i}.t + c_6.t + \max._b(c_5.t, c_{16}.t) + c_5.t + c_{12}.t + c_{15}.t + AE_{6.o}.t$
$\mathbf{AE_7}$	$S_{AE_7}.t = AE_{7.i}.t + 3\,c_{11}.t + 2\,c_7.t + AE_{7.o}.t$
$\mathbf{AE_8}$	$S_{AE_8}.t = AE_{8.i}.t + c_{11}.t + c_6.t + 3\,c_7.t + AE_{8.o}.t$

The maximum total computation times to serve the aperiodic events $AE_1,...,AE_8$ for sequential execution of the parallelizable components are shown in Table D.1. The branching operations have been shown with $max._b$ operator. These equations also show the maximum computational complexity to serve the corresponding aperiodic events.

D.2.1 Sensor Fusion System (SFS) Running on Dedicated Single Computing Node

If a dedicated computing node is assigned to execute these computing components as shown in Fig. D.1, the waiting time of each component in the queue is zero. The service time of each event (i.e., the execution of a component upon its arrival in the queue) is equal to the corresponding execution time only. In this operational scenario, components arrive in the queue only if the queue is empty. The addition of extra computing node will not reduce the service times of the aperiodic events. Therefore, the computing power of this single computing node should be adequate to guarantee that Ts ≤ Te for each computing component. As in Appendix B, the maximum utilization factor of this computing system will be 27.67% to avoid overlapping in the DEVR model. Moreover, all the components used to estimate the maximum service time of each aperiodic event will not always be in operation. They will come in operation only if some specific conditions are met which depend upon the sensing environment. Therefore, the utilization factor of this computing system will be lower (may be much lower) than 27.67%.

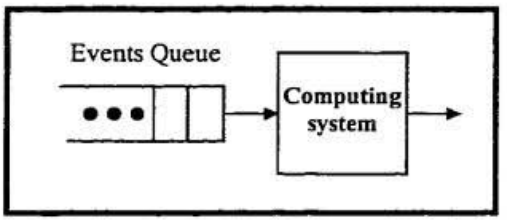


Figure D.1: Single node based computing system.

D.2.2 Multiple SFSs Running on Single Computing Node

To increase the utilization factor, multiple SFS can be implemented on the same computing system. As explained in section 4.6, DEVR models of multiple SFSs can be interlaced to increase the utilization factor. For this example problem, it has been shown in section B.6 (to specify sensing sequence) that there are idle periods in the DEVR model of this example SFS. In

this operational scenario, interlacing the DEVR model of another SFS with the DEVR model of this example SFS can increase the utilization factor. Due to the interlacing of the DEVR models, the components will not wait in the queue of the computing node to be executed resulting in zero waiting time. Therefore, the computing power of the single computing node should be adequate enough to guarantee that $Ts \leq Te$ for each computing component of both the SFSs. The addition of extra computing nodes will not increase the system performance in terms of service time.

All the computing components used to calculate the maximum total execution times to serve these aperiodic events will only come in operation if certain operating conditions of the sensing environment are met. Therefore, even in the busy period of the DEVR model the computing node will not be always busy. Therefore, multiple DEVR models can be implemented on a single node to share the same busy period. In this scheme, the service time of each computing component (i.e., the execution time and the waiting time) will be random due to the randomness in the waiting time which is a function of the arrival rate of the component in the queue as explained in Section 4.3. Under this operational scenario, the addition of extra node will decrease the waiting time resulting in reduced service time. The request of the execution of the components will come to single queue and multiple nodes will use the same queue as shown in Fig. D.2. The task coordination unit will make sure that the components belonging to the same SFS do not go into execution on both the processors.

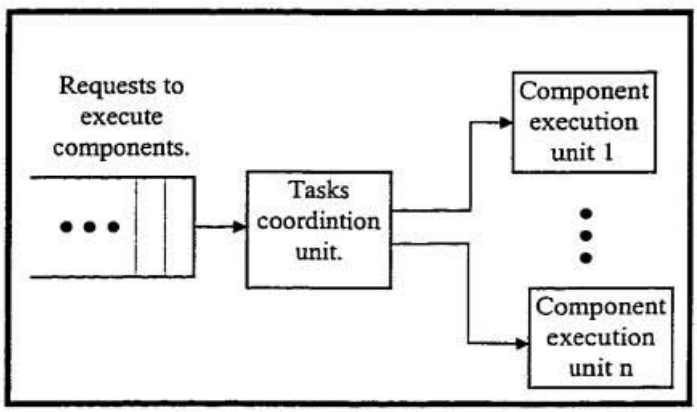


Figure D.2: Multiple nodes serve requests from the same queue resulting in reduced waiting time.

D.3 The Architecture of the Computing System while Parallelizable Components are Executed in Parallel Fashion

The maximum total computation times to serve the aperiodic events $AE_1,...,AE_8$ for parallel executions of the parallelizable components are shown in Table D.2. The parallel operations have been shown with $max._b$ operator.

Table D.2: The maximum total computation times to serve the aperiodic events for the parallel executions of parallelizable components.

Aperiodic events	The maximum computation times
$\mathbf{AE_1}$	$S_{AE_1}.t = AE_{1.i}.t + max._p [\{2c_7.t + c_{13}.t\}, \{max._b(c_5.t, c_{16}.t) + c_{13}.t\}] + AE_{1.o}.t$
$\mathbf{AE_2}$	$S_{AE_2}.t = AE_{2.i}.t + max._p\{c_1.t, max._b(c_{13}.t, c_{16}.t)\} + max._p(c_5.t, c_{12}.t\} + AE_{2.0}.t$
$\mathbf{AE_3}$	$S_{AE_3}.t = AE_{3.i}.t + max._p\{c_{15}.t, max._b(c_{13}.t, 3c_{11}.t)\} + max._p(c_5.t, c_{12}.t) + AE_{3.o}.t$
$\mathbf{AE_4}$	$S_{AE_4}.t = AE_{4.i}.t + c_8.t + max.p(c_2.t, 4\ c_{11}.t) + c_4.t + AE_{4.o}.t$
$\mathbf{AE_5}$	$S_{AE_5}.t = AE_{5.i}.t + c_8.t + max._p(c_2.t, 4\ c_{11}.t) + max._b(c_4.t, c_{13}.t, c_5.t) + AE_{5.o}.t$
$\mathbf{AE_6}$	$S_{AE_6}.t = AE_{6.i}.t + max._p\{c_6.t, max._b(c_5.t, c_{16}.t)\} + max._p(c_5.t, c_{12}.t, c_{15}.t) + AE_{6.o}.t$
$\mathbf{AE_7}$	$S_{AE_7}.t = AE_{7.i}.t + max._p(3\ c_{11}.t, 2\ c_7.t) + AE_{7.o}.t$
$\mathbf{AE_8}$	$S_{AE_8}.t = AE_{8.i}.t + c_{11}.t + max._p(c_6.t, 3\ c_7.t) + AE_{8.o}.t$

In these execution sequences to serve the aperiodic events $AE_1,...,AE_8$, the maximum level of parallelization is 3. Therefore, the underlying computing system may have three independent nodes with separate queues to achieve higher temporal performance as shown in Fig. D.3. In this architecture the parallelizable components will be sent to separate queues. For example, to serve the aperiodic event AE_6 the components c_3, c_{12}, and c_{15} will be executed simultaneously by the three independent nodes 1, 2, and 3 respectively thus reducing total computational time.

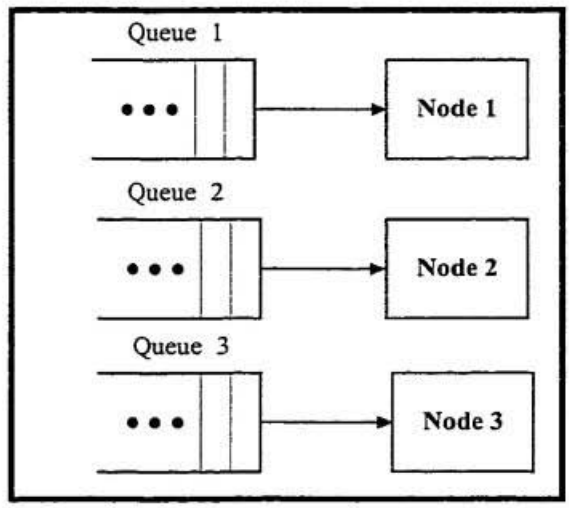


Figure D.3: The three parallels independent computing nodes to execute parallelizable components parallely.

The utilization factor of this multi-node based parallel architecture will be lower than single-node based architecture. Because, during the execution of sequentially executable components the two of three nodes will remain idle. If it is assumed that the execution time of each computing component is equal, the state of operations of each these of three computing nodes to serve the aperiodic event AE_6 when all the components come in operation are shown in Fig. D.4. Here, the node 3 and node 2 remain idle during 66% and 33% of the total service time of AE_6 respectively.

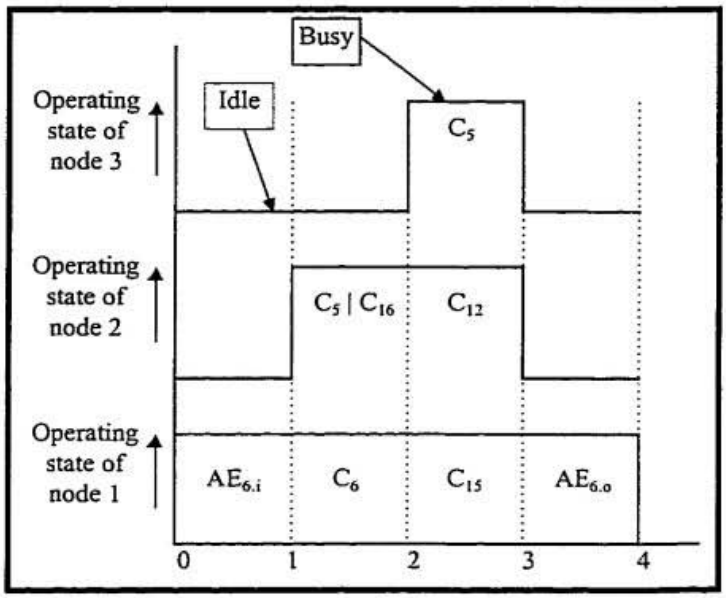


Figure D.4: The operating states of different nodes to serve the aperiodic event AE_6.

Different approaches to enhance the overall utilization factor as explained in the previous section for the single node based system can be applied here too.

The modern computing processors utilize pipelined architecture to exploit instruction level parallelism in order to reduce the required clock cycles per instruction (CPI). This attribute of these modern processors contributes to the randomness to the execution time of an instruction. The instruction execution time depends not only upon the computational complexity of the instruction alone, but also upon the instructions already in different stages of executions in the pipeline [63]. This has been explained in the following sub-section.

D.4 The Randomness in the Execution Time of a Computing Component on Pipelined Architecture

To understand this problem a closer look can be taken to the pipeline structure of a particular class of architecture. Due to the superior performance of reduced instruction set computer (RISC) over compound instruction set computer (CISC), the effect of pipeline structure of the MIPS R4000 processor family on the randomness of the instruction execution times has been studied here. The R4000 uses a eight-stage pipeline structure and sometimes this is called superpipeline as shown in Fig. D.5 [63]. The pipe stages are labeled and their detailed functions are described in the following text.

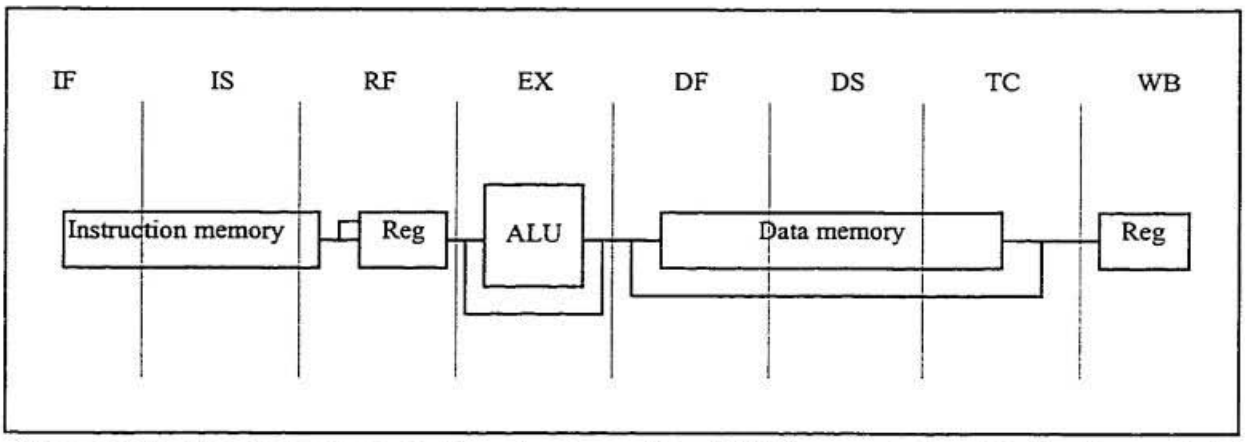


Figure D.5: The eight-stage pipeline structure of the R4000 uses pipelined instruction and data caches [42].

The function of each stage is as follows:

1. IF : First half of the instruction fetch.
2. IS : Second half of instruction fetch.
3. RF : Instruction decode and register fetch.
4. EX : Execution.
5. DF : First half of the data fetch.
6. DS : Second half of the data fetch.
7. TC : Tag check.
8. WB : Write back.

The dependence of the execution time of an instruction on other instructions already at different stages of executions in the pipeline is shown in Fig. D.6.

	Clock number								
Instruction number	1	2	3	4	5	6	7	8	9
LW R1	IF	IS	RF	EX	DF	DS	TC	WB	
ADD R2,R1		IF	IS	RF	stall	stall	EX	DF	DS
SUB R3,R1			IF	IS	stall	stall	RF	EX	DF
OR R4,R1				IF	stall	stall	IS	RF	EX

Figure D.6: The dependence of the execution time of an instruction on the instructions already in execution in the pipeline [63].

The variations of clock cycles per instruction (CPI) for the 10 SPEC92 benchmarks [63] are shown in Table D.3. The variations of these pipelined CPIs in graphical form are shown in the Fig.D.7.

Table D.3: The randomness of total pipelined CPI and the contributions of the four major sources of stalls are shown [63].

Benchmark	Pipeline CPI	Load stalls	Branch stalls	Floating point (FP) result stalls	FP structural stalls
compress	1.20	0.14	0.06	0.00	0.00
eqntott	1.88	0.27	0.61	0.00	0.00
espresso	1.42	0.07	0.35	0.00	0.00
gcc	1.56	0.13	0.43	0.00	0.00
li	1.64	0.18	0.46	0.00	0.00
Integer average(IA)	1.54	0.16	0.38	0.00	0.00
doduc	2.84	0.01	0.22	1.39	0.22
mdjdp2	2.66	0.01	0.31	1.20	0.15
ear	2.17	0.00	0.46	0.59	0.12
hydro2d	2.53	0.00	0.62	0.75	0.17
su2cor	2.18	0.02	0.07	0.84	0.26
FP average	2.48	0.01	0.33	0.95	0.18
Overall average(OA)	2.00	0.10	0.36	0.46	0.09

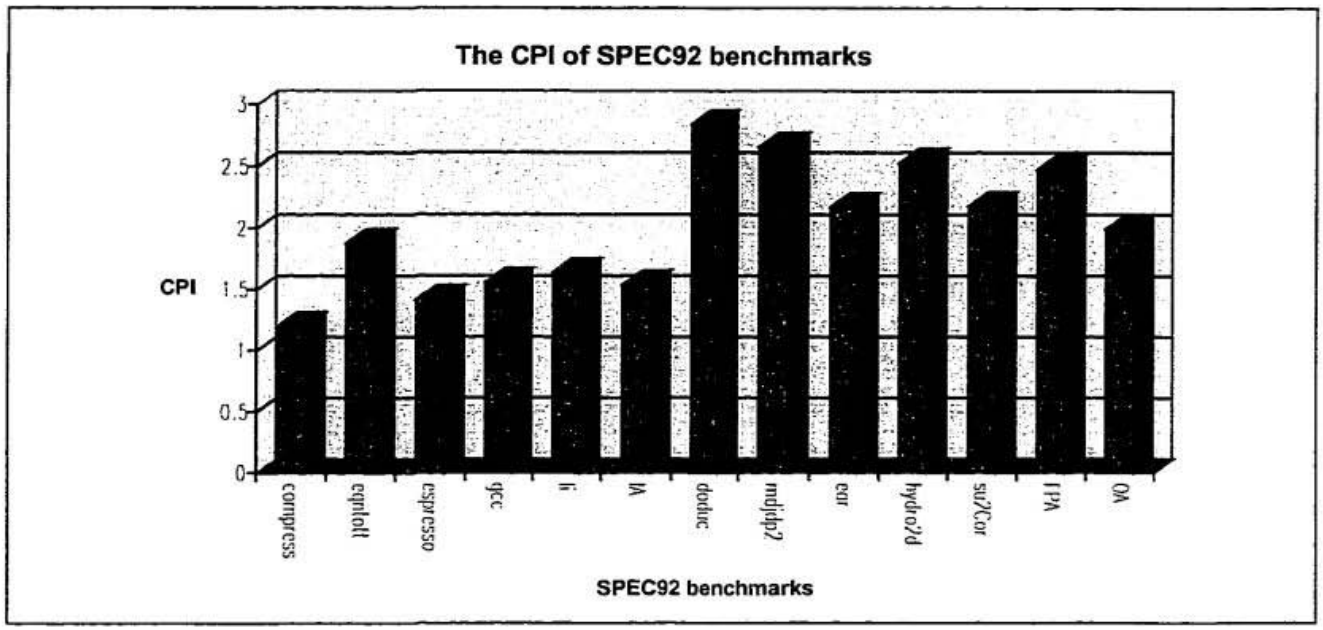


Figure D.7 The variations of the MIPS R4000's pipelined CPI of SPEC92 benchmarks

Table D.4: The statistics of the variations of CPI of SPEC92 benchmarks

Maximum	Minimum	Average	Variance	(Max.-Min)*100/Min
2.84	1.20	2.00	.21	164%

The statistics of the randomness of the MIPS pipelined CPI of SPEC92 benchmarks is shown in Table D.4. In this study, it appears that the maximum clock cycles per instruction (CPI) is 164% higher than the minimum CPI. Therefore, it is not reasonable to depend on the CPI values to calculate the time required executing a computing component on modern pipelined processing units. It is recommended to run the computing component on those CPUs to estimate the time required for its execution. If this point is not considered in the implementation phase of the SFS, the system may suffer setback to satisfy the temporal specifications of the DEVR and DEVS models. This setback will result in the development of less reliable sensor fusion system. Therefore, the consideration of this limitation (from temporal point of view) of modern central processing units will help us realize reliable sensor fusion system.

D.5 Randomness in Execution Time of a Computing Component on Hierarchical Memory Architecture

To understand the contribution of the hierarchical memory architecture of modern computing system on the randomness of the execution times of the computing components it can be assumed that the system has cache (internal and external), main memory and disk storage. These four levels of memory hierarchy with increasing capacity and decreasing speed as shown in Fig.D.8 [62]. The data transfers sequence between successive levels of memory hierarchy during the execution of the computing components is shown in Fig. D.9.

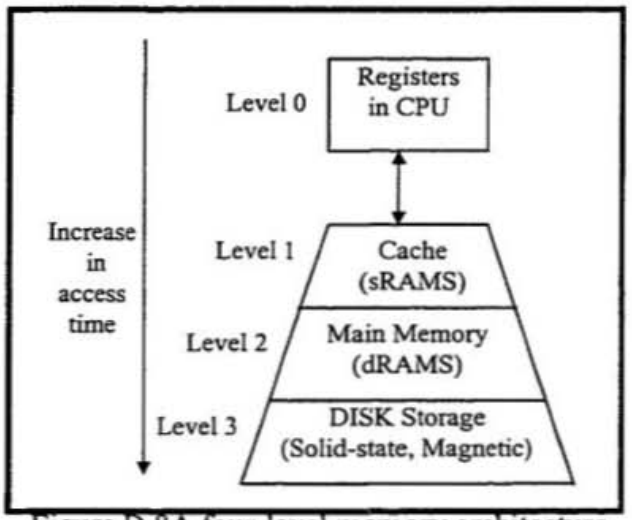


Figure D.8A four-level memory architecture

To quantify the effect of the presence of required data in particular level of memory on the execution time of an instruction, the typical values of access times of different levels of memory are shown in Table D.5 [62].

Table D.5: Typical values of access times of different levels of memory.

Memory Level	Access time
CPU Registers	A few clock cycles
Cache (SRAMs)	25 ns
Main Memory	70 ns
Disk array	5 ms

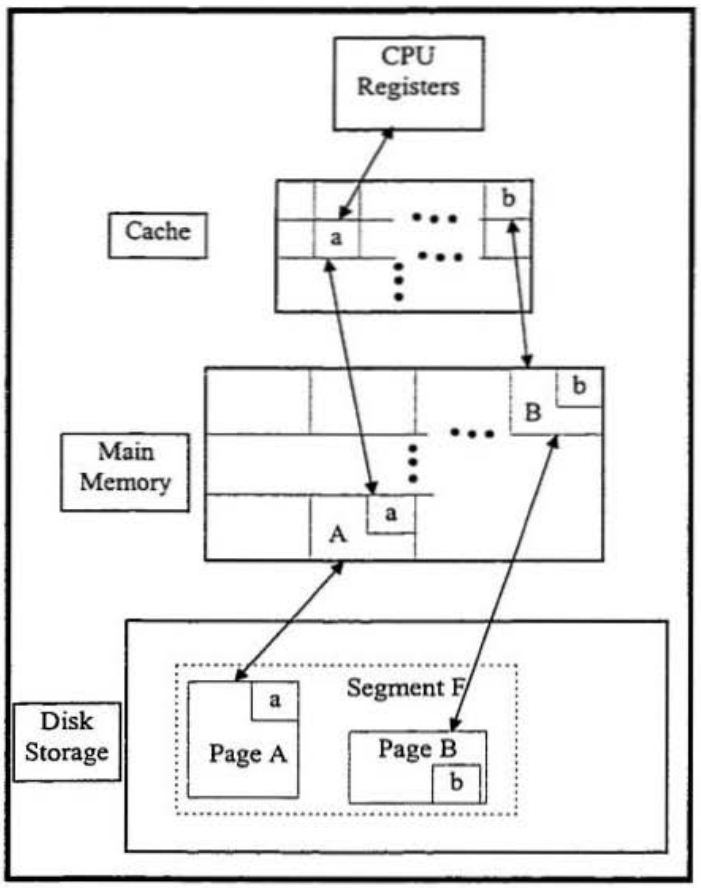


Figure D.9:Data transfer between adjacent levels

The data access time in disk is $4x10^5$ times higher than that in Cache. Therefore, the availability of data required for the execution of a particular computing component at different memory levels will contribute highly to the total execution time of that component.

The memory reference patterns for the execution of the computing components are caused by the following locality properties [62],[63]:

1. Temporal locality
2. Spatial locality
3: Sequential locality

The spatial and sequential localities depend upon the memory access patterns of a particular component. The temporal locality not only depends upon the memory access pattern of the particular component, but also upon the data distribution at different levels made by the executions of the past computing components. Therefore, the memory access time during the execution of a particular component not only depends upon the data access pattern of the code of that component alone, but also on the data access pattern of the previously executed components. The sequence of execution of different computing components to serve different aperiodic events is random due to the randomness of the arrival of aperiodic events. This randomness of data distribution at different memory levels by the previously executed components will cause randomness in the execution time of a computing component. To explain the problem, the effect of different memory reference patterns on the execution times of a computing component can be considered as shown in Table D.6.

Table D.6: The effect of data distribution on the memory access time

Scenarios	Data available in Cache	Data available in Memory	Data available in Disk	Total memory access time in
1	40%	30%	20%	≈100 ms
2	30%	40%	30%	≈150 ms
3	20%	50%	30%	≈150 ms
4	60%	40%	0%	4300 ns
5	40%	60%	0%	5200 ns

From the data shown in Table D.6, it is evident that there is a potential of high degree of randomness in the execution time of a computing component due to the randomness of the initial data distribution at different memory levels by the previously executed components. This randomness in execution time may result in failure of temporal correctness of the execution of the computing components to serve different events. As a result the reliability of the system will suffer.

One of the solutions of this problem may be achieved by flushing the different levels of memory at the beginning of the execution of each computing component. This will result in a memory reference pattern of each computing component independent of data distribution caused by the previously executed components.

In this study, the rationale for the embedded computing architecture to implement the example SFS has been explained. The quantitative measure of the sources of randomness of the component execution times on modern computing hardware has been provided. This finding will help ensure temporal correctness in execution time.

Appendix E

Improvement of the Reliability and the Required Overhead for the Incorporation of Hardware Fault-Tolerance in the Example SFS

E.1 Introduction

The example sensor fusion system has seven sensors and these sensors are sensed sequentially. If it is assumed that this SFS is implemented on a single processor based system and all the sensors are sensed using the same analog channel, the high-level hardware configuration is as shown in

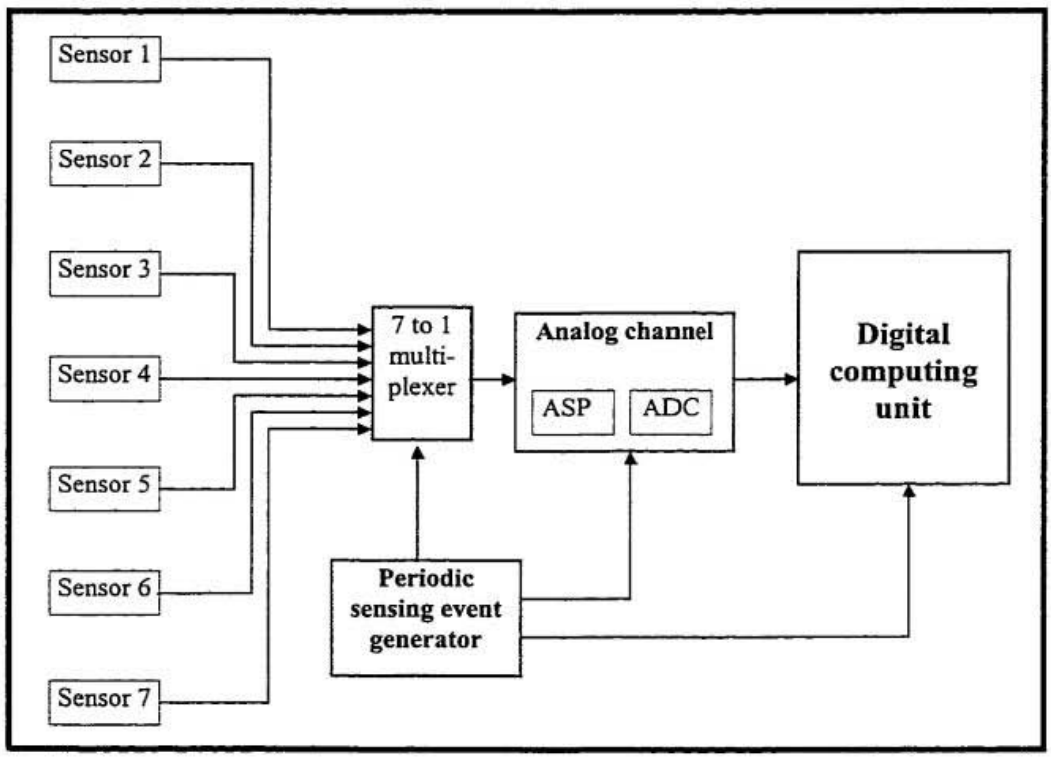


Figure E.1: The hardware configuration of the example sensor fusion system.

Fig.E.1. This study will focus on the improvement of reliability and the required overhead for the incorporation of hardware fault-tolerance. The fault-tolerance examined here only addresses the issues of the failure of sensors. It has been assumed that other components will function properly.

E.2 Hardware, Energy, and Space Overhead to Incorporate Fault-Tolerance

The required overhead to implement fault-tolerance depends upon the techniques used to detect fault sensors. The voting and estimation are the most prominent fault detection techniques as explained in Chapter 4.

E.2.1 Overhead to Incorporate Fault-Tolerance Using Voting Technique Based Faults Detections

If the sensor fault detection scheme is implemented using majority voting technique, for triple modular redundancy (TMR) the system will require *7x3=21* sensors and 7 voters. The TMR uses three identical sensors with a majority voter to determine the output as shown in Fig.E.2. All these sensors should be in operation resulting in at least *200%* increase in power to drive the sensor suit. These extra *14* sensors, *7* voters and the required extra energy source to keep them operational will also require more space. As a result the material and operation cost will rise. The nature of the applications and enhancement of system reliability may justify this cost.

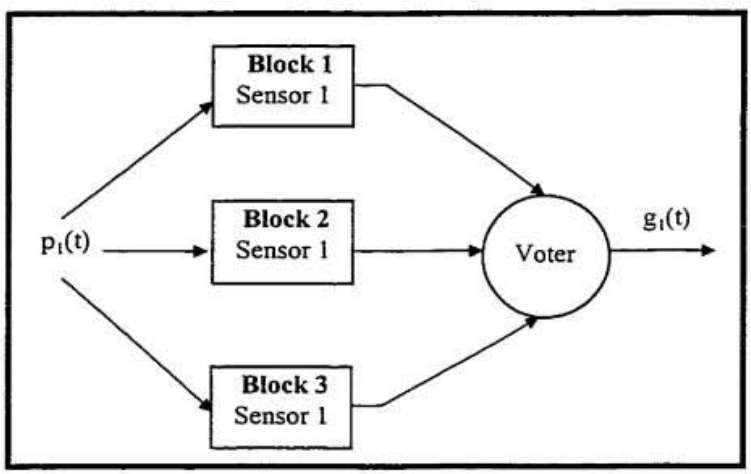


Figure E.2: Triple modular redundancy implementation of sensor 1

E.2.2 Overhead to Incorporate Fault-Tolerance Using Estimation Technique Based Faults Detections

In fault-tolerant sensor system using estimation technique based fault detection scheme as shown in Fig. E.3, it is not necessary to keep all the redundant sensors operational. This will reduce the energy overhead in comparison to hardware based TMR technique. Triple modular redundancy based on estimation technique will require the same number of sensors (7x3=21). The hardware based TMR system failed after the failure of one sensor as shown in the state diagram as shown in Fig.E.4 [36], but in estimation based technique the system will be functioning as long as one fault-free sensor is available. As a result, it appears that the estimation based technique has the potential to have better reliability profile than voting based fault detection system.

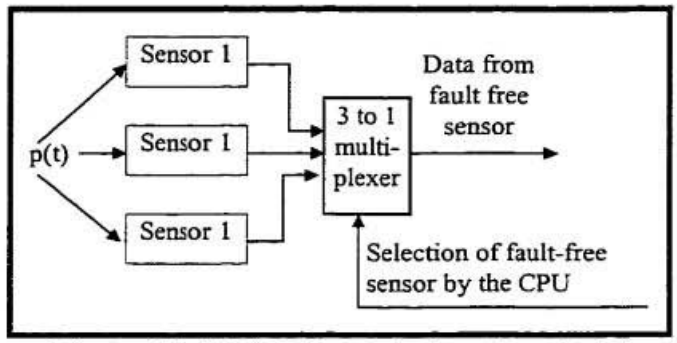


Figure E.3: Estimation technique based triple modular redundant sensor system.

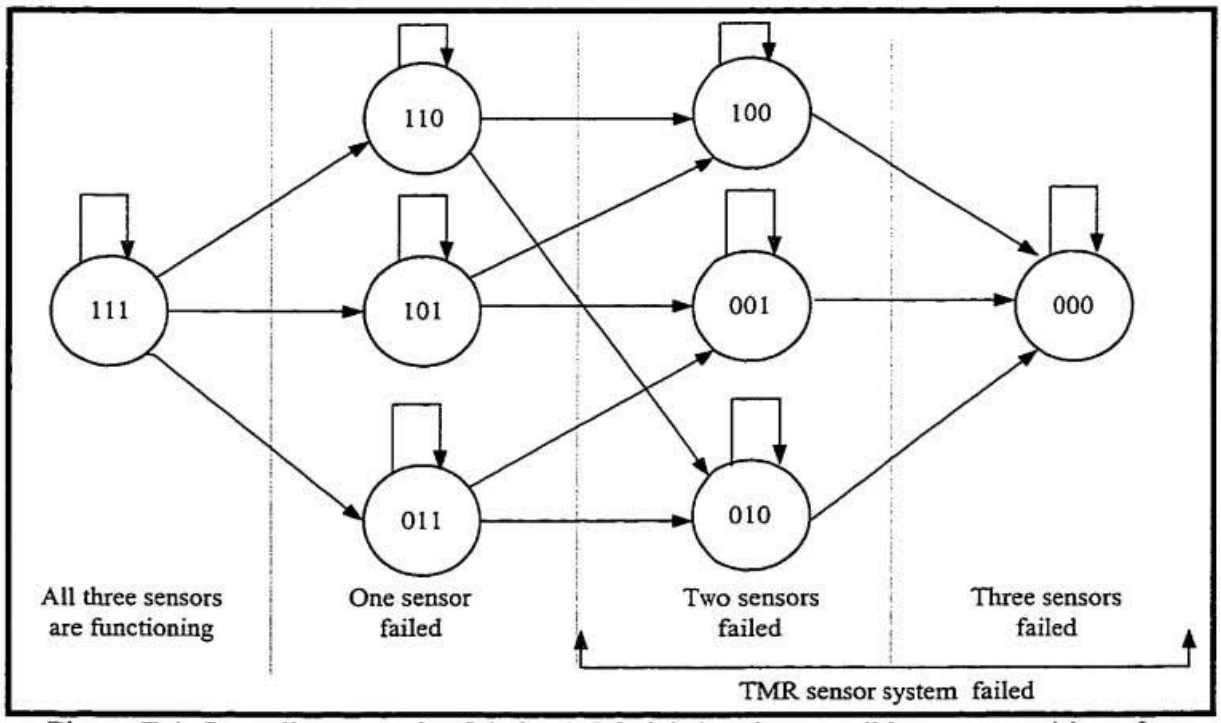


Figure E.4: State diagram using Markov's Model showing possible state transitions for TMR system.

E.3 Reliability Profile of a Fault Tolerant Sensor System Using Voting Based Fault Detection Scheme

If $R_1(t)$ is the reliability of the first sensor and the other two sensors to support triple modular redundancy have same reliability, the reliability of the first TMR sensor system is as follows

$$R_{1_{TMR}}(t) = 3R_1^{\,2}(t) - 2R_1^{\,3}(t) \qquad \text{(E.1)}$$

In a TMR sensor system, as long as two of the three sensors are functioning correctly, the sensor system will perform correctly. In the above equation the reliability of the voter has been ignored. The reliability profile of TMR sensor system in comparison to the reliability of a single sensor is shown in Fig. E.5. From this figure it is evident that the reliability of the TMR system will be higher only if the reliability of a single sensor is more than 50%. In realistic sense, most of the sensor's reliability is more than 80% for reasonable lifetime. Therefore, this approach has the potential to increase system's reliability at the cost of extra overhead. The use of 4-modular redundancy instead of triple modular redundancy will require only one additional sensor. The reliability of 4-modular sensor system is as follows

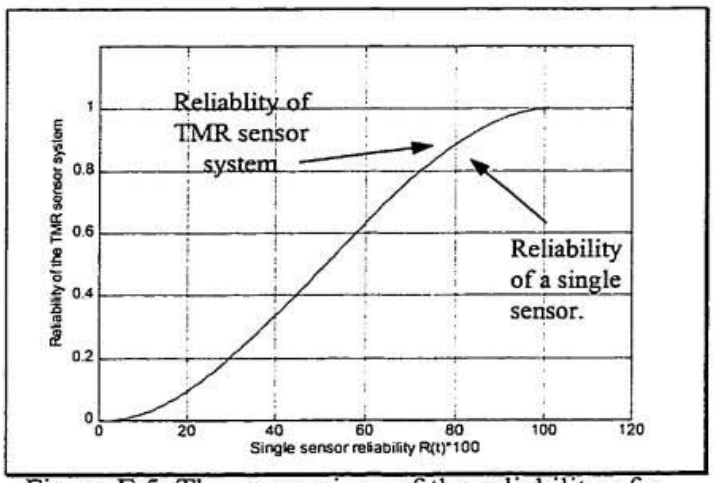


Figure E.5: The comparison of the reliability of a TMR system consisting of the three identical sensor modules with the reliability of a single sensor.

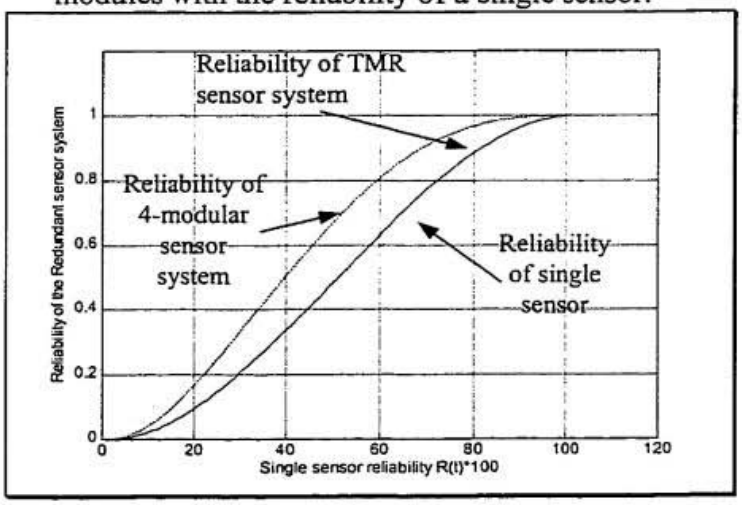


Figure E.6: The comparison of the reliability profile of 4-modular sensor system with those of TMR sensor system and single sensor.

$$R_{4-\text{mod}}(t) = 3R^4(t) - 8R^3(t) + 6R^2(t) \qquad \text{(E.2)}$$

The comparison of the reliability of 4-modular redundancy sensor system with the reliabilities of TMR and single sensor is shown in Fig. E.6. From this graph, it appears that the 4-modular redundancy has much better reliability profile than that of TMR system at the cost of one additional sensor.

E.4 Reliability Profile of a Fault Tolerant Sensor System Using Estimation Based Fault Detection Scheme

A triple modular sensor system as shown in Fig. E.3 will function properly as long as one fault-free sensor is available. It appears that these three sensors are functioning as independent signal sources; the signal will be available as long as one source is functioning correctly. The following relation measures the reliability of such system:

$$R(t) = 1 - (1 - R_1(t))^3 \tag{E.3}$$

Here, the failure of the multiplexer has been ignored and it has been assumed that all the sensors have the same reliability. The comparison of the reliability profiles of TMR sensor system using estimation based fault detection technique with those of TMR sensor system using voting technique to detect faults, and the single sensor is shown in Fig. E.7. From these reliabilities, it appears that fault-tolerant system using estimation based fault detection technique has much better reliability profile than that of sensor system using voting technique based fault detection scheme. Moreover, the reliability of the TMR sensor system using estimation technique is always higher than that of single sensor.

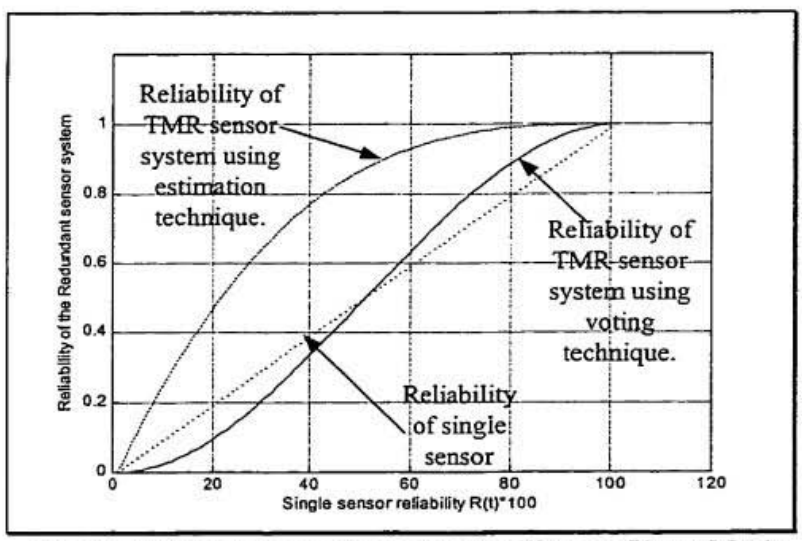


Figure E.7: The comparison of reliability profiles of fault-tolerant sensor using voting technique based fault detection technique with those of fault-tolerant sensor using estimation based fault detection technique, and single sensor.

E.5 The Comparisons of the Reliability Profiles of Different Fault-Tolerant Sensor Systems and Single Sensor

The graph shown in Fig. E.8 gives a comparison of the reliability profiles of fault-tolerant systems having different levels of redundancy using voting and estimation techniques for fault detection.

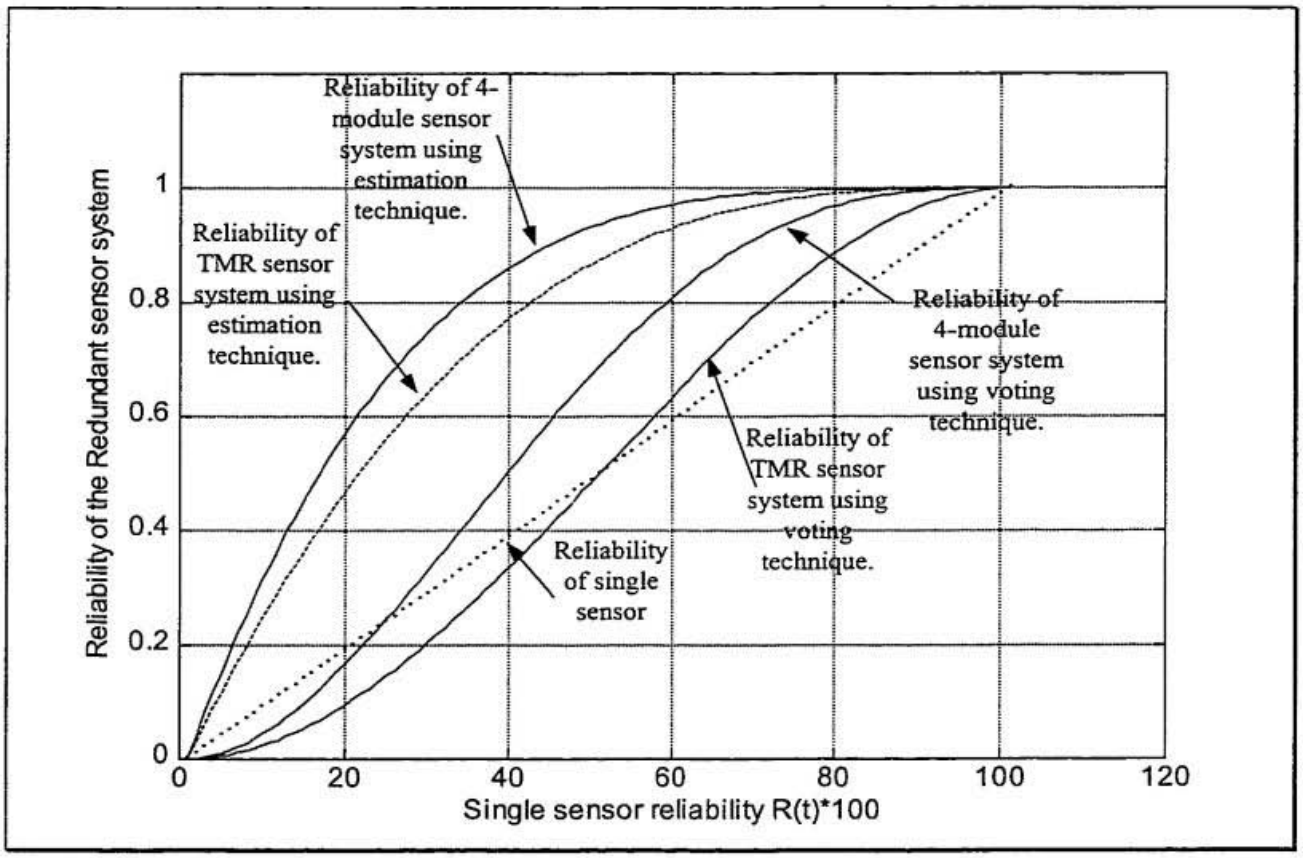


Figure E.8: The comparison of reliability profiles of fault-tolerant sensor system having different levels of redundancy using voting and estimation techniques.

From this graph, it appears that the estimation based technique continues to show better performance than the voting based technique. It should be noted that unlike voting technique the estimation based technique does not require separate voter for each sensor module. If the probability of failure of this voter were brought under consideration, the estimation based fault-tolerant system would show much better performance than voting based technique.

For more quantitative comparisons of the reliability profiles of fault-tolerant sensor systems using voting and estimation techniques, the ratios of the reliability profiles of TMR and 4-module sensor systems using estimation to those of TMR and 4-modular redundant systems using voting technique are shown in Fig.E.9. For better comparisons in the region of high reliability values, the ratio profiles have been shown for the reliability of more than 30% of a single sensor.

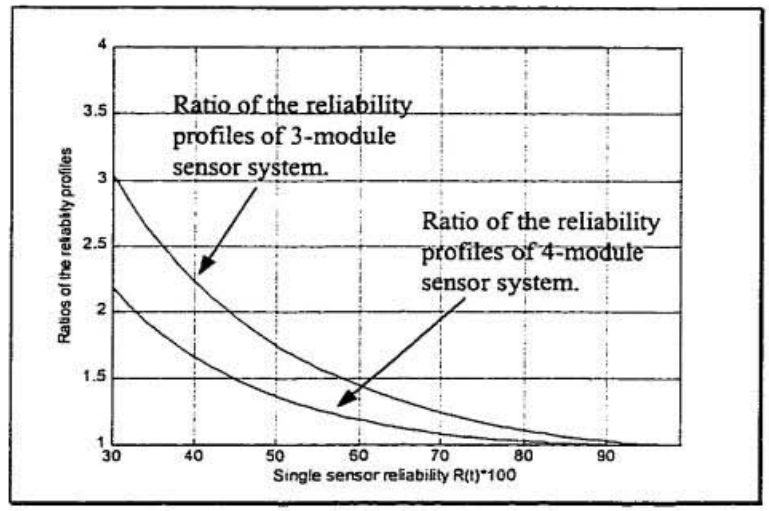


Figure .E.9: The ratios of reliability profile of fault-tolerant sensor system using estimation and voting techniques for fault detection.

E.6 The Reliability Profile of the Example Sensor Fusion System at Different Levels of Fusion

The sensor fusion system specified in appendix A will partially fail due to the failure of any terminal events (AE_1, AE_8, AE_5) as shown in the DEVR model in appendix B.

It is also necessary to quantify the probability that data will be provided at different levels of fusion: data fusion, feature fusion, and decision fusion by the supporting sensors. This measure will enable the designer to measure the system performance to extend the functionality of already designed sensor fusion system. The reliability profiles of the terminals will be first studied in the following subsections. Then this study will be continued to quantify the reliability profiles of different levels of fusion. This study is limited to the failure of the supporting sensors only.

E.6.1 The Reliability Profile of Terminal Event AE_1

A fault tree shown in Fig depicts the relationship of the failure of the sensors to the failure of AE1. E.10. The reliability of the aperiodic event AE_1 in terms of the reliabilities of the supporting sensors 1 and 2 is as follows:

$$R_{AE_1}(t) = 1 - (1 - R_1(t))(1 - R_2(t)) \qquad \text{(E.4)}$$

The reliability profile of the AE_1 at different levels of redundancy in the supporting sensor suite is shown in Fig. E.11 with the assumption that all the sensors have same reliabilities.

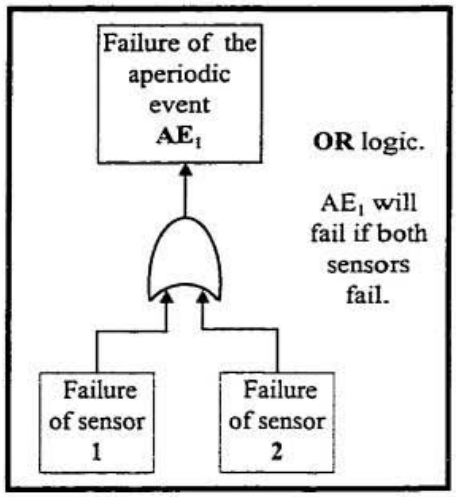


Figure E.10: Fault tree of AE_1 in relation to the failure of the supporting sensors 1 and 2.

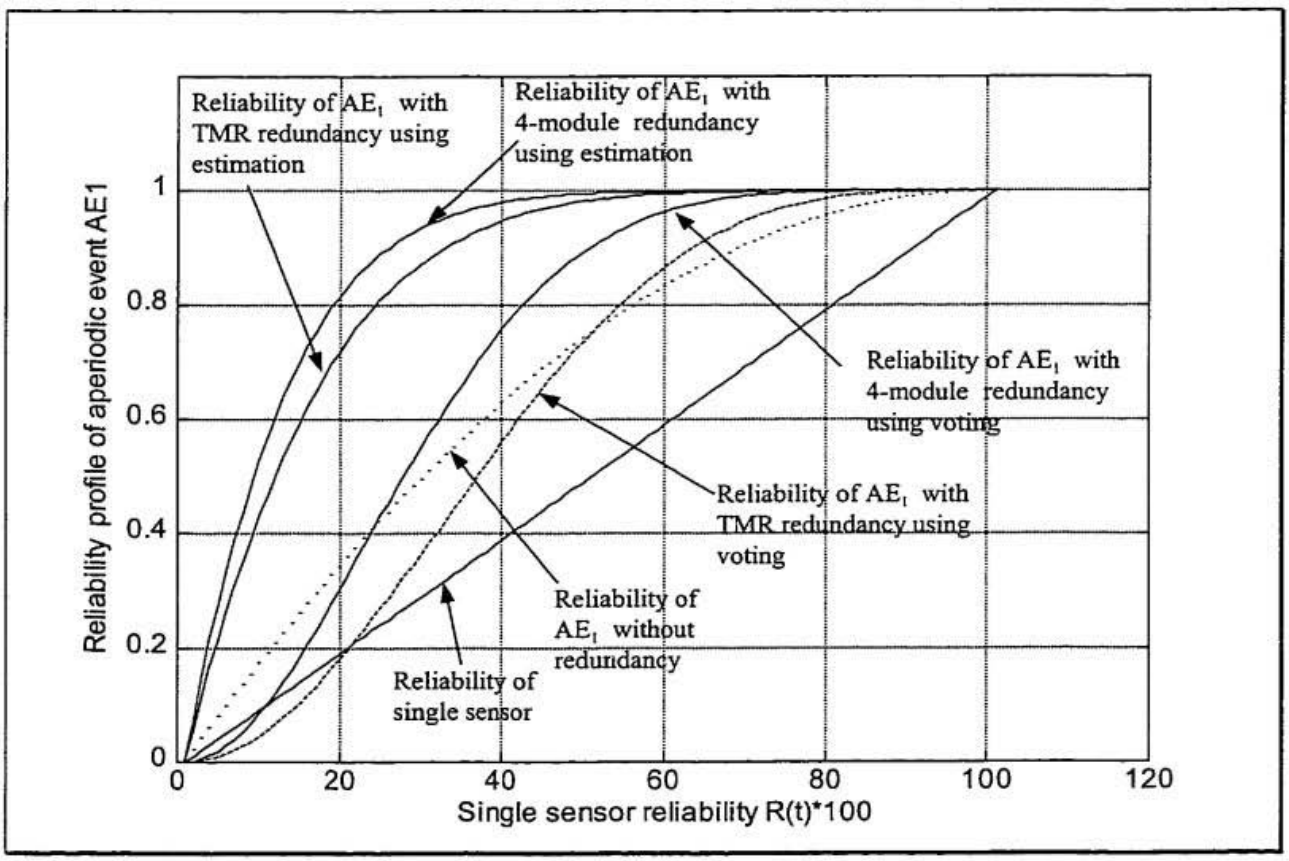


Figure E.11: The reliability profile of the aperiodic event AE_1

E.6.2 The Reliability Profile of Terminal Event AE_5

The fault-tree of the aperiodic event AE_5 in relation to the failure of the supporting sensors is shown in Fig. E.12. To support data for the service of AE_5, sensors 4 and 5 play redundant role, while sensors 4 and 5 are complementary to sensors 6 and 7. The reliability of the aperiodic event AE_5 in terms of the reliabilities of the supporting sensors 4, 5, 6, and 2 is as follows:

$$R_{AE_5}(t) = (1-(1-R_4(t)(1-R_5(t))R_6(t)R_7(t) \quad \text{(E.5)}$$

The reliability profile of AE_5 is shown in Fig.E.13.

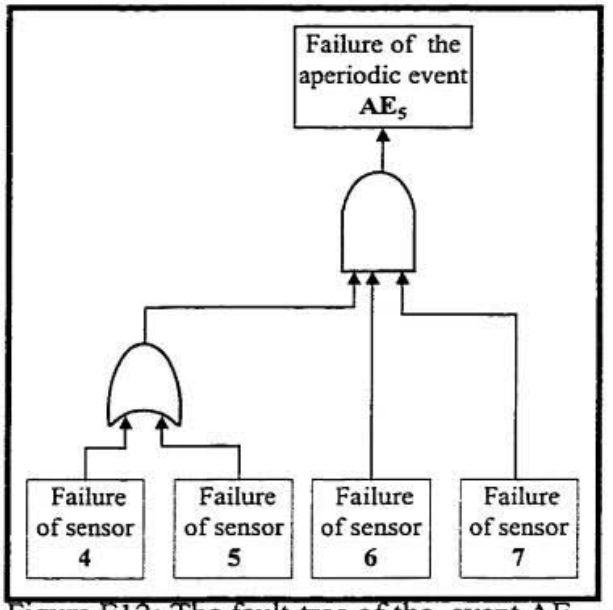


Figure E12: The fault-tree of the event AE_5.

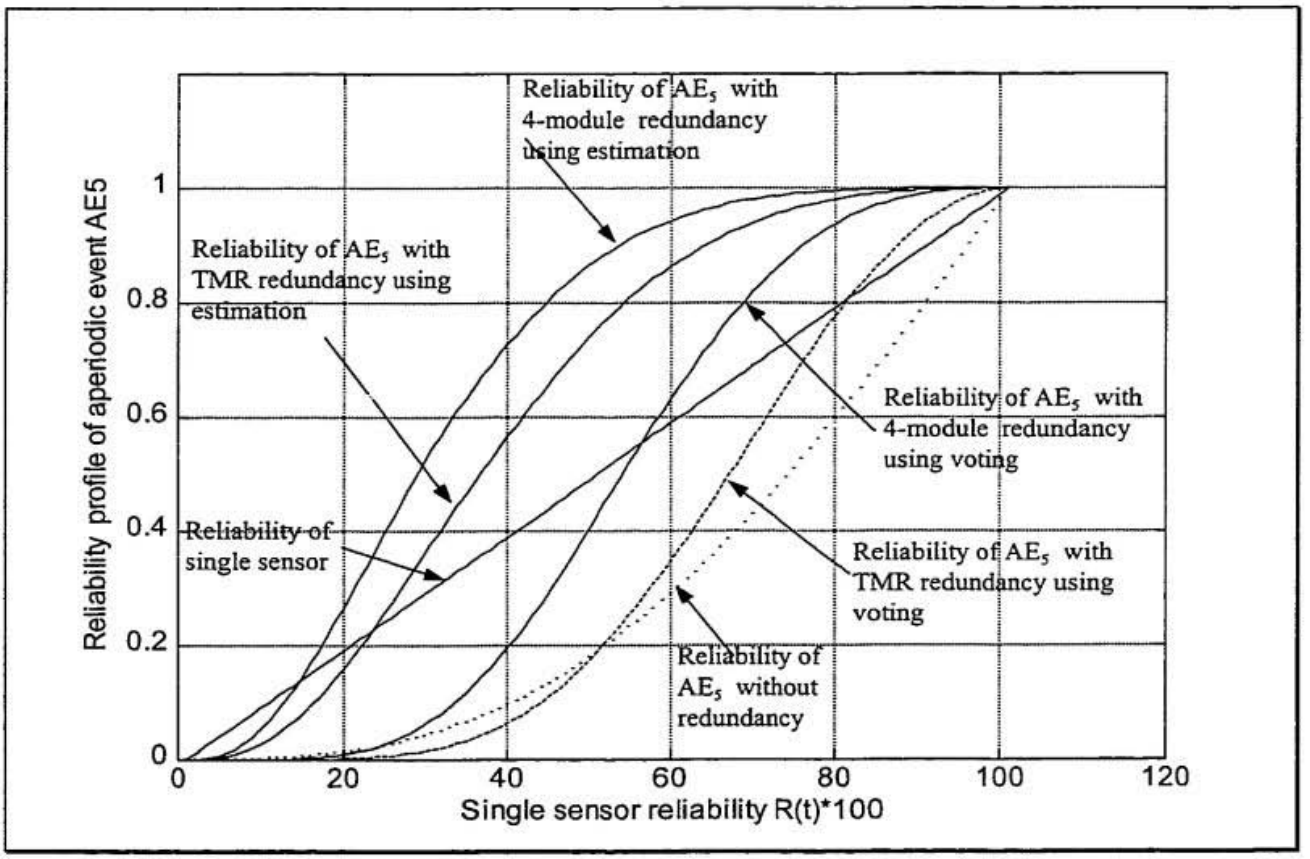


Figure E.13: The reliability profile of the aperiodic event AE_5.

E.6.3 The Reliability Profile of Terminal Event AE_8

The fault-tree of the aperiodic event AE_8 in relation to the failure of the supporting sensors is shown in Fig. E.14.

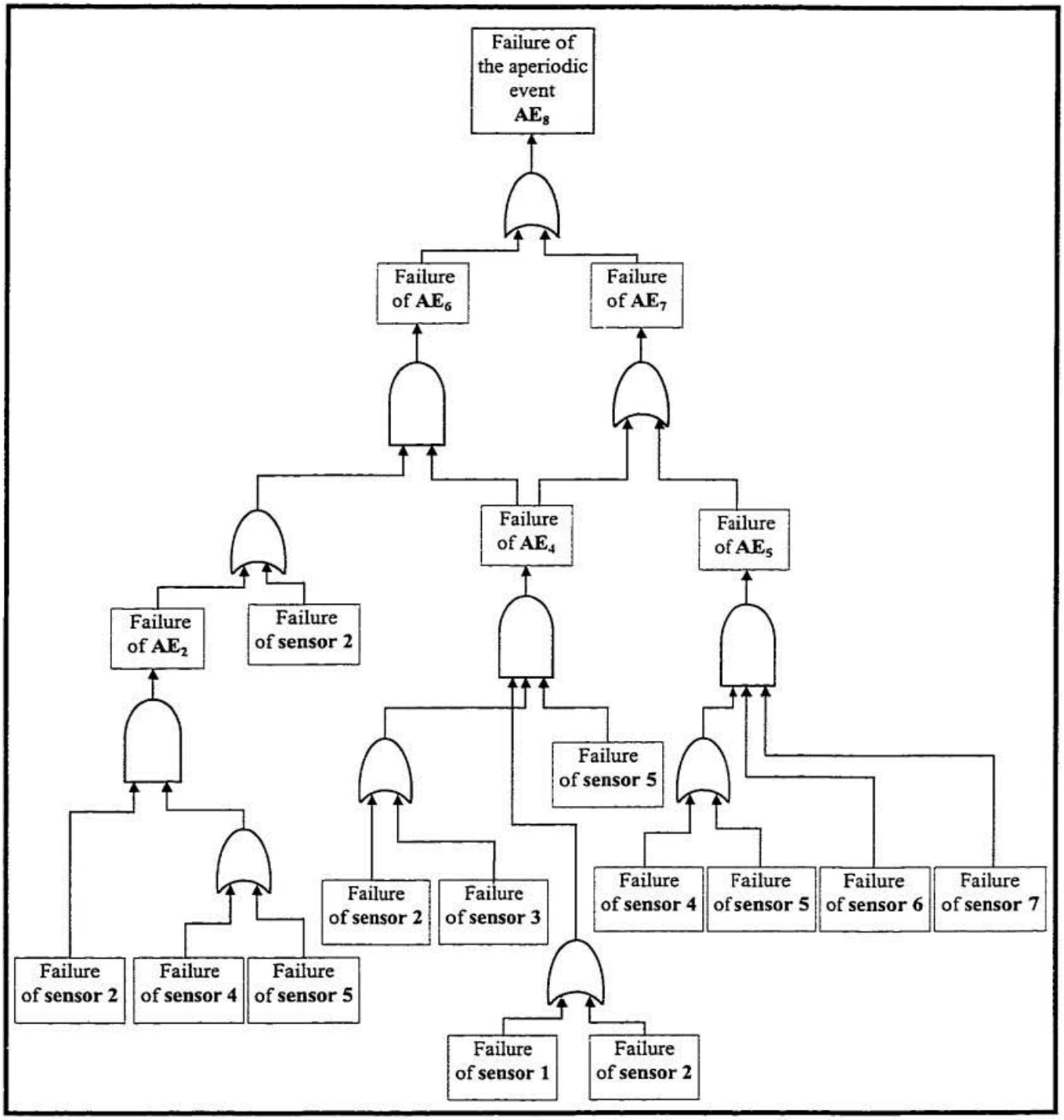


Figure E.14: The fault-tree of the failure of aperiodic event AE_8.

The derivation of the relation to estimate the reliability of the aperiodic event AE_8 in terms of the reliabilities of the supporting sensors with the assumption that all the sensors have same reliabilities is shown below.

$$R_{AE_2}(t) = R(t)(1-Q^2(t)), \quad Q(t) \text{ is the unreliability} \tag{E.6}$$

$$R_{AE_4}(t) = \{1-Q^2(t)\}\{1-Q^2(t)\}R(t) \tag{E.7}$$

$$R_{AE_5}(t) = \{1-Q^2(t)\}R(t)R(t) \tag{E.8}$$

$$R_{AE_6}(t) = [1-\{1-R(t)(1-Q^2(t)\}Q(t)][\{1-Q^2(t)\}\{1-Q^2(t)\}R(t)] \tag{E.9}$$

$$R_{AE_7}(t) = [1-\{1-Q^2(t)\}\{1-Q^2(t)\}R(t)][1-\{1-Q^2(t)\}R(t)R(t)\}] \tag{E.10}$$

$$R_{AE_8} = [1-[1-\{1-R(t)(1-Q^2(t)\}Q(t)][\{1-Q^2(t)\}\{1-Q^2(t)\}R(t)]][1-[1-\{1-Q^2(t)\}\{1-Q^2(t)\}R(t)][1-\{1-Q^2(t)\}R(t)R(t)\}]] \tag{E.11}$$

The reliability profile of the AE_8 at different levels of redundancy in the supporting sensor suite is shown in Fig. E.15.

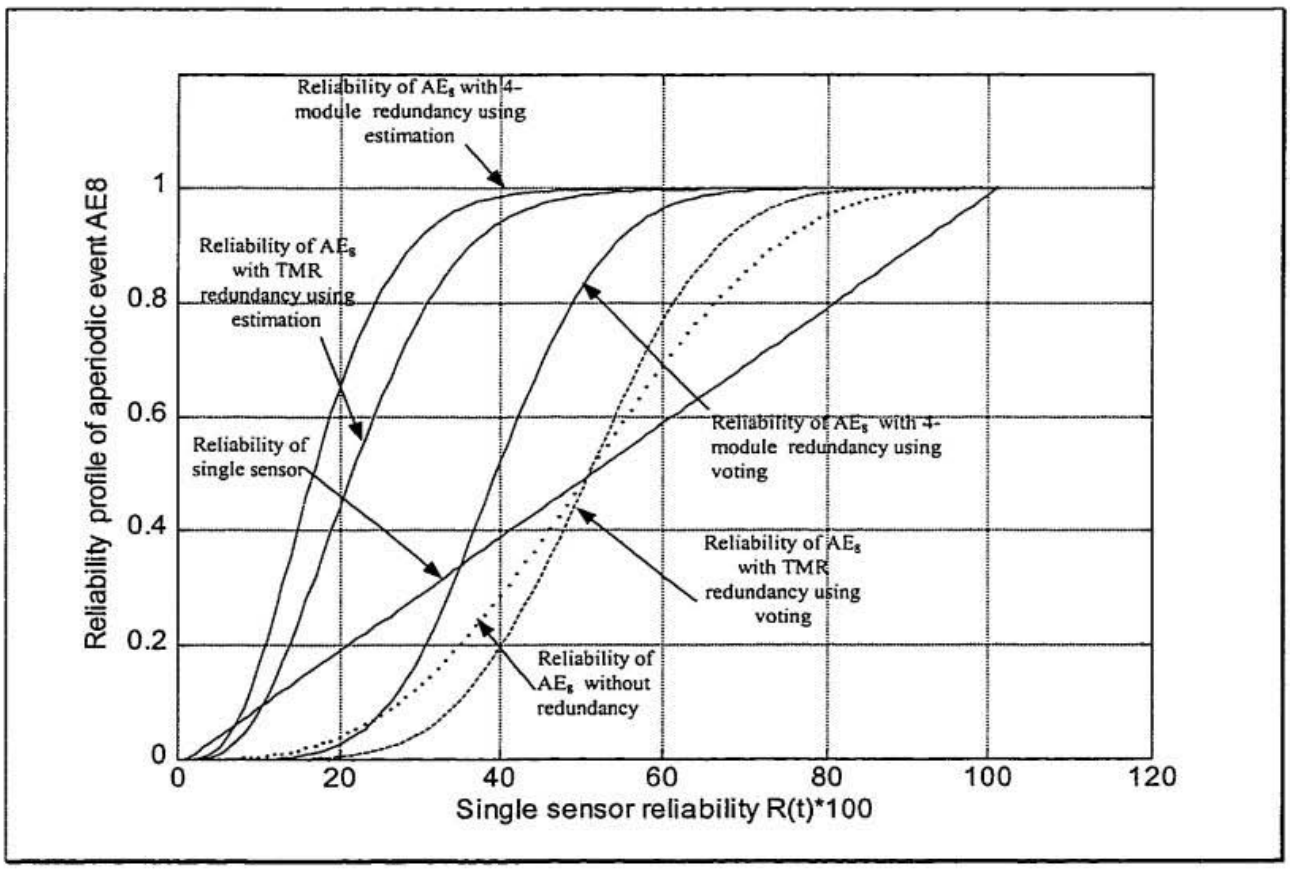


Figure E.15: The reliability profile of the aperiodic event AE_8.

E.6.4 The Reliability Profile of the Data Fusion with Event AE_2

The data level fusion of data from sensors 2, 4, and 5 is performed by the service of event AE_2. The fault-tree of the aperiodic event AE_2 is shown in Fig.E.16. The reliability of providing data from the sensors to generate the event AE_2 is shown below.

$$R_{AE_2}(t) = R(t)(1-Q^2(t)\ ) \qquad \text{(E.12)}$$

The profile of reliable data supply to the aperiodic event AE_2 is shown in Fig.E.17.

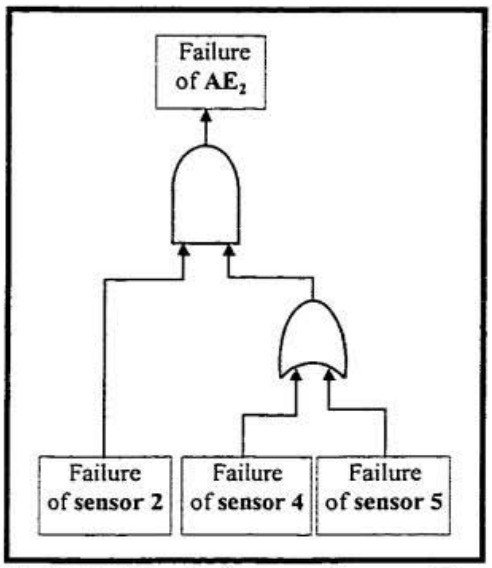


Fig.E.16: The fault-tree of event AE_2.

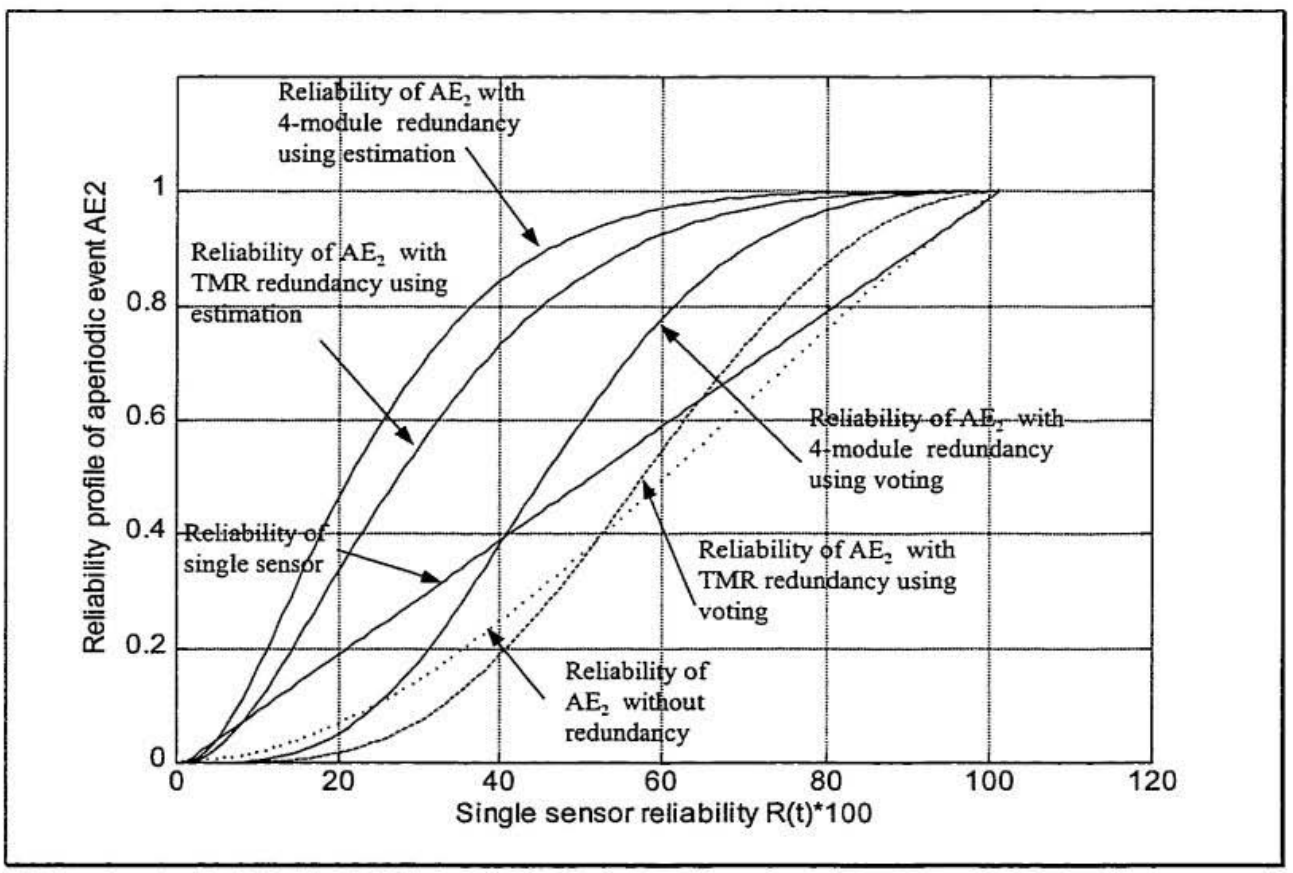


Figure E.17: The reliability profile of the aperiodic event AE_2.

E.6.5 The Reliability Profile of the Data Fusion with Event AE_3 and AE_4

The reliability profiles of AE_3 and that of sensor 2 are the same, because AE_3 is provided data only by that sensor. The fault-tree of the aperiodic event AE_4 is shown in Fig.E.18. The following relation measures the readability of the event AE4:

$$R_{AE_4}(t) = \{1 - Q^2(t)\}\{1 - Q^2(t)\}R(t) \quad \text{(E.13)}$$

The graph shown in Fig.E.19 gives the reliability profile of event AE_4.

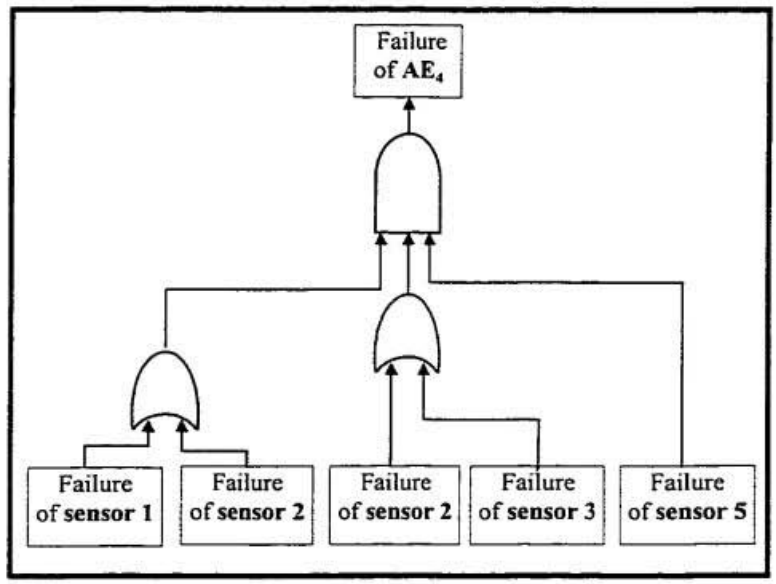


Figure E.18: The fault-tree of the event AE_4.

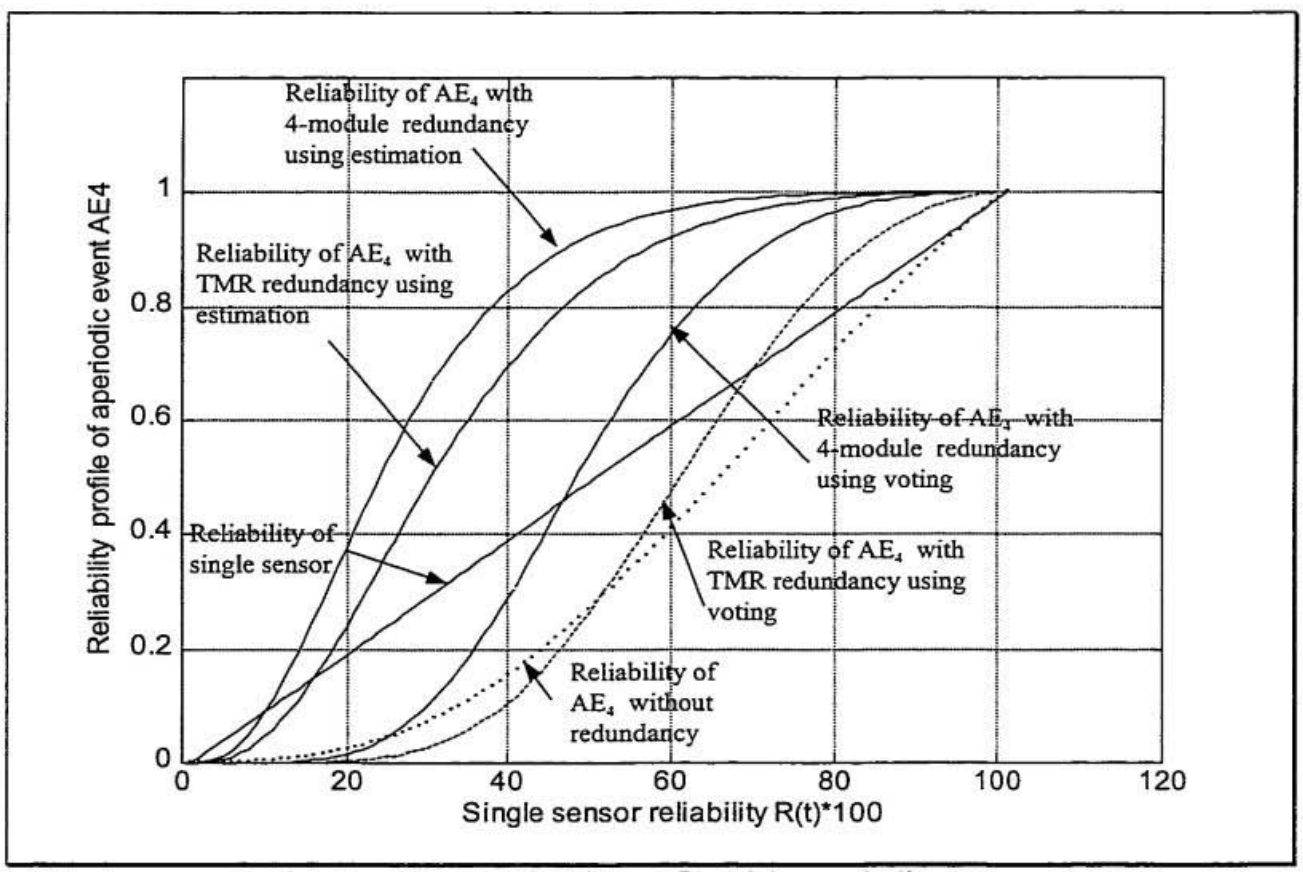


Figure E.19: The reliability profile of the aperiodic event AE_4.

E.6.6 The Reliability Profile of the Feature Fusion with Event AE_6

The failures of the events AE_2 and AE_4 contribute to the failure of AE_6. The fault-tree of the aperiodic event AE_6 is shown in Fig.E.20. The reliability that data will be provided to AE_6 by the sensors is calculated by the following equations.

$$R_{AE_6}(t) = (1-(1-R_{AE_2}(t))(1-R(t))R_{AE_4}(t) \qquad \text{(E.14)}$$

$$R_{AE_6}(t) = [1-\{1-R(t)(1-Q^2(t)\}Q(t)][\{1-Q^2(t)\}\{1-Q^2(t)\}R(t)] \qquad \text{(E.15)}$$

The reliability profile of the event AE_6 is shown in Fig.E.21.

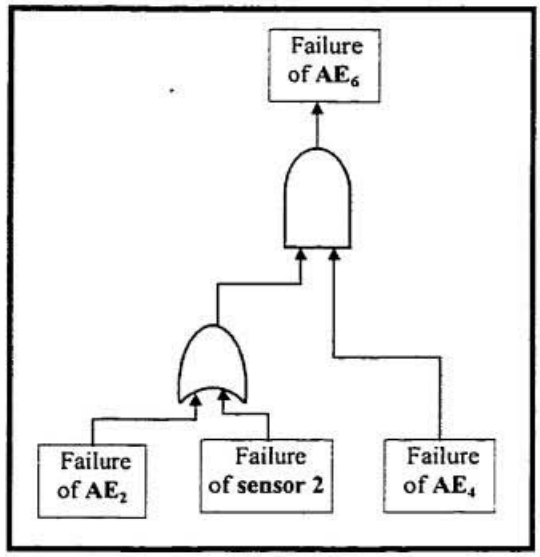


Figure E.20: The fault-tree of the aperiodic event AE_6.

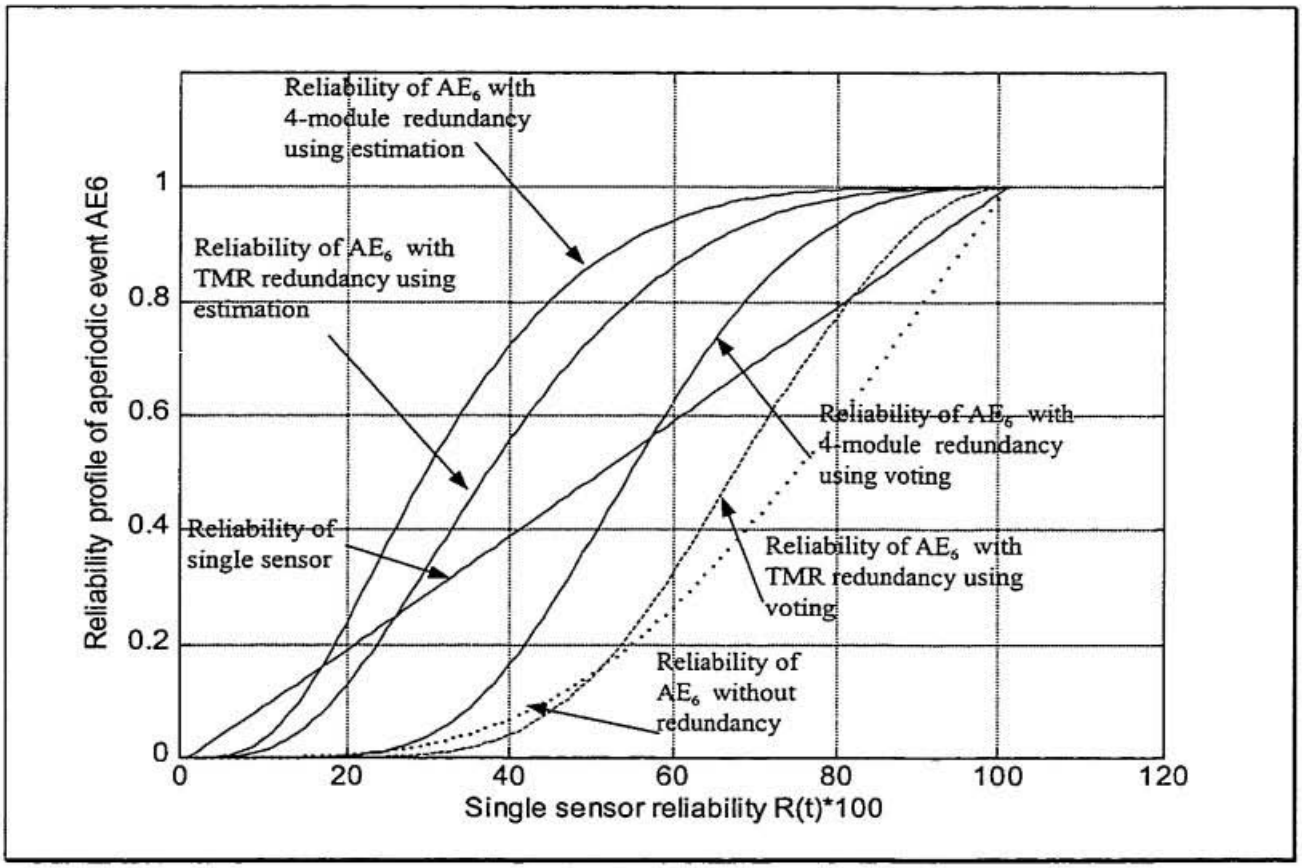


Figure E.21: The reliability profile of the aperiodic event AE_6.

E.6.7 The Reliability Profile of the Data Fusion with Event AE_7

The output of the aperiodic events AE_4 and AE_5 are redundant. The failure of both of these events will result in failure of the event AE_7. The fault-tree of AE_7 in terms of the failure of AE_4 and AE_5 is shown in Fig.E.22. The reliability that data will be provided to AE_7 by the sensors is estimated by the following equations.

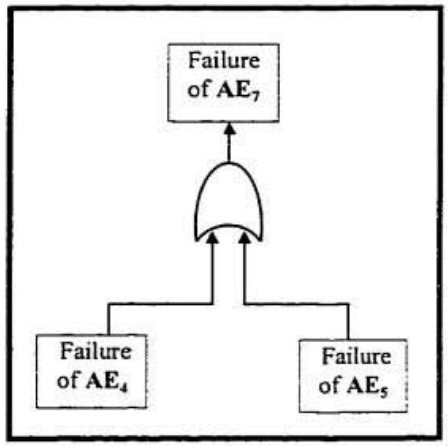


Figure A.E.22: The fault-tree of the aperiodic event AE_7.

$$R_{AE_7}(t) = 1-(1-R_{AE_4}(t))(1-R_{AE_5}(t)) \quad \text{(E.16)}$$

$$R_{AE_7}(t) = [1-\{1-Q^2(t)\}\{1-Q^2(t)\}R(t)][1-\{1-Q^2(t)\}R(t)R(t)\}] \quad \text{(E.17)}$$

The reliability profile of this aperiodic event is shown in Fig. E.23.

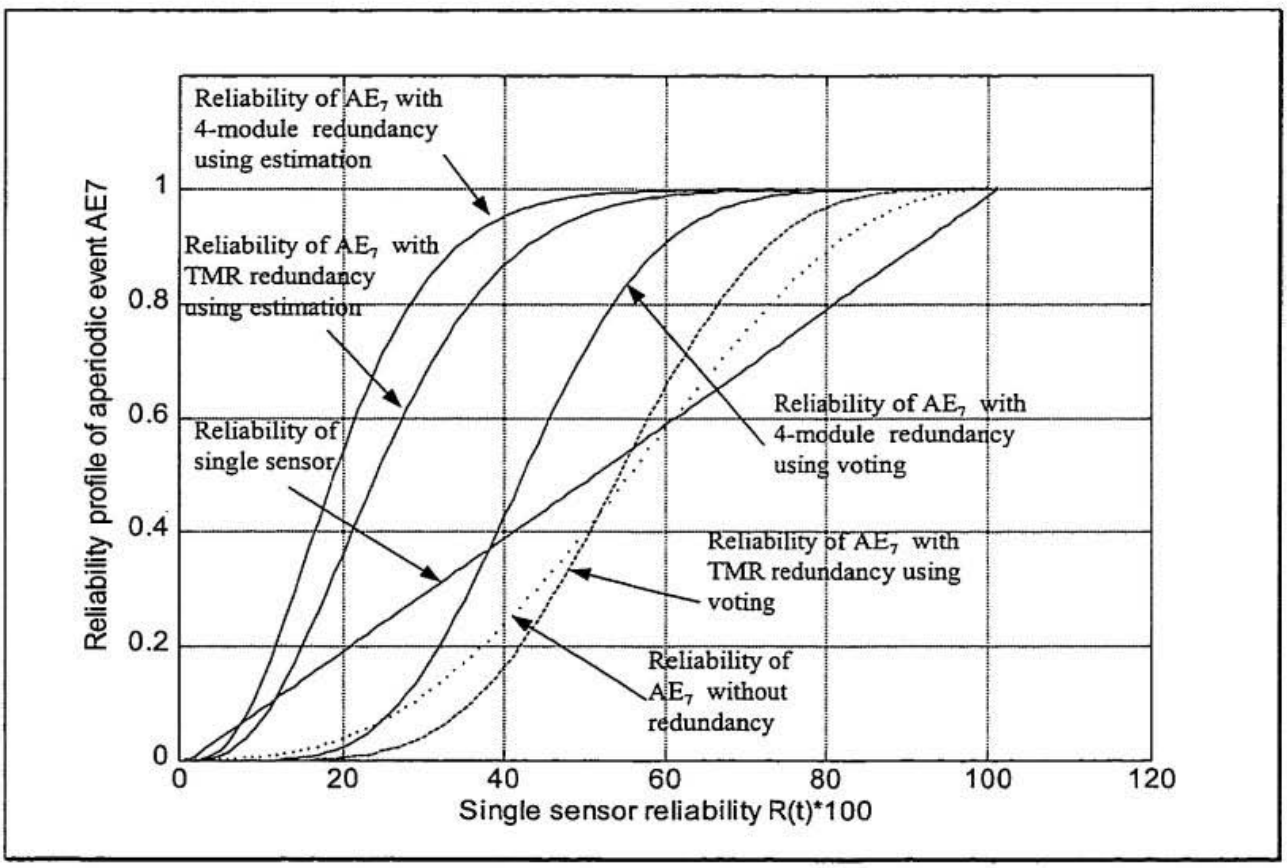


Figure E.23: The reliability profile of the aperiodic event AE_7.

E.7 Temporal Overhead to Manage Redundancy

The temporal overhead to manage redundancy depends upon the technique used for sensors faults detection.

In voting technique, if fault occurs during the data acquisition time (DAT) as explained in section 5.2.1.2, fault should be cleared instantly. The time required to detect and switch the faulty sensor with a fault-free one will be very negligible. During this fault clearance period data will be lost and these data may be recovered using parallel sensing technique as explained in chapter 6 and the time of this recovery will be small in comparison to the total allocated time for periodic process as shown in Table A.3. In this case, the lower limits of the service times of the periodic processes should be extended by this data recovery time. If faults occur during TCAD, service time of the aperiodic events and WT, the data recovery is not required and voter may have the potential to operate autonomously resulting in no temporal overhead on the DEVR model. Therefore, in this operational scenario, DAT should include enough time to recovery data lost during fault clearing period when fault occurs during DAT.

In estimation technique, at the end of every data acquisition session estimation algorithm should be run on these acquired data to make sure that the sensor is fault-free. This sensing sequence is explained in Fig. E.24. In this sensing scenario, the total allocated times for periodic processes as shown in Table A.3 should include enough time for at least acquiescing data twice and running the estimation algorithm twice to accommodate at least one failure between two successive periodic events. This requirement will create tremendous temporal overheard on the DEVR model. The estimation technique has the potential to show better reliability profile in comparison to voting technique, but this is at the cost of extra temporal overhead on each periodic process.

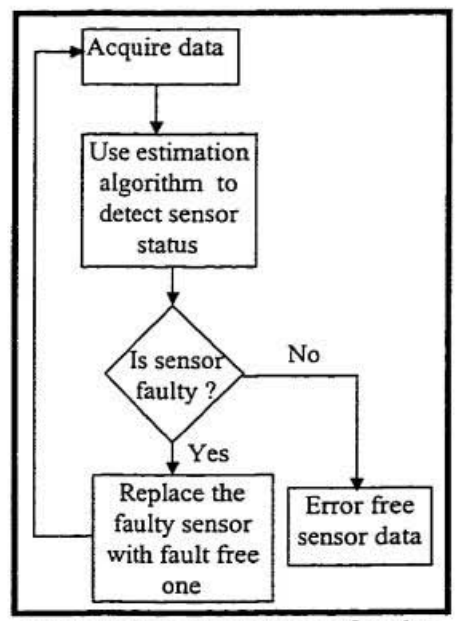


Figure E.24: Sequence of tasks to acquire fault-free data while estimation technique is used to detect faulty sensors.

Appendix F

Detection of Sensor Faults in Multi-sensori System by Simulation

F.1 Introduction

In this simulation study, four test signals have been composed by the combination of different harmonic components. The Fourier series representation of the formation of these test signals is given by

$$f(t) = a_0 + \sum_{n=1}^{\infty} a_n \cos(n\omega t) + \sum_{n=1}^{\infty} b_n \sin(n\omega t) \qquad \text{(F.1)}$$

where t being an independent variable represents time, $\omega = \frac{2\pi}{T}$ is the first harmonic, n is the number of harmonics, and a_n and b_n are the amplitudes of the harmonics $n\omega$.

The specification of this simulation is shown in Table F.1. The study of local means and variances of these test signals at fault free condition will be followed by the study of these parameters at different fault conditions. The study will examine the effects of different instances of occurrence and different frequencies of transient faults on local statistics. The variations of these local statistics with the variation of window size and location will also be investigated. The permanent faults causing +ve and -ve saturation will only be addressed here.

Table F.1: The specifications of the simulating environment.

Signal duration	Signal amplitude range	Sampling frequency	Sensor noise
100 ms	1 to 9V	50 kilo samples/sec	Normally distributed random variable

F.2.1 The Characteristics of the First Test Signal

The following relation gives the formation of this test signal:

$$f_1(t) = 5 + 2\sin((2\pi 20)t) + 2\cos((2\pi 20)t) \qquad \text{(F.2)}$$

The equivalent physical signal, sensor signal with noise, dynamic local mean and variance of the sensor signal are shown in Fig. F.1, Fig.F.2, Fig.F.3, and Fig. F.4 respectively.

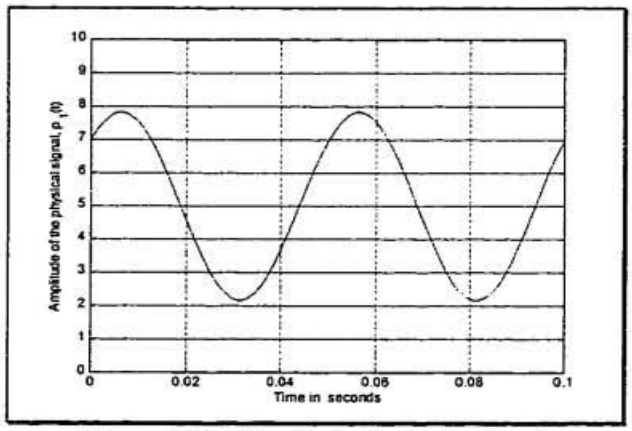


Figure F.1: The first physical signal.

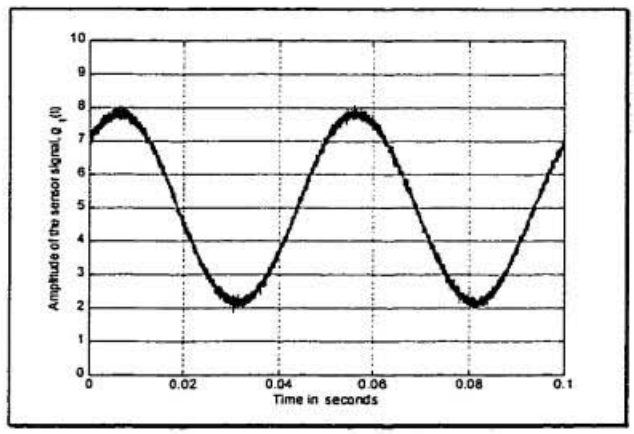


Figure F.2: The first sensor signal with noise.

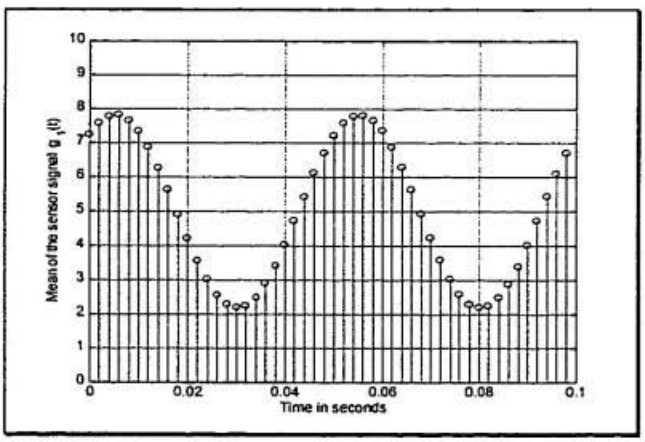


Figure F.3: The local means of the first signal.

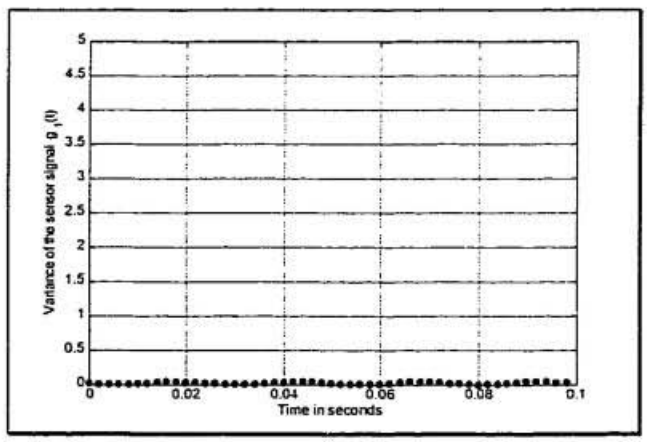


Figure F.4: The local variances of the signal.

The domains of the physical signal, sensor signal, local means and variances of this sensor signal are given in Table F.2.

Table F.2: The statistics of the first test signal.

Signals	Upper bound	Lower bound
$p_1(t)$	7.82	2.17
$g_1(t)$	8.09	1.85
Local means	7.82	2.18
Local variances	0.0517	0.0057

F.2.2 The Characteristics of the Second Test Signal

The formation of the second test signal is given by the following relation

$$f_2(t) = 5 + \sin((2\pi 20)t) + 1.5\cos((2\pi 20)t) + 1.5\sin((4\pi 20)t) \quad \text{(F.3)}$$

The equivalent physical signal, sensor signal with noise, the local means and variances of the sensor signal are shown in Fig. F.5, Fig.F.6, Fig.F.7, and Fig. F.8 respectively.

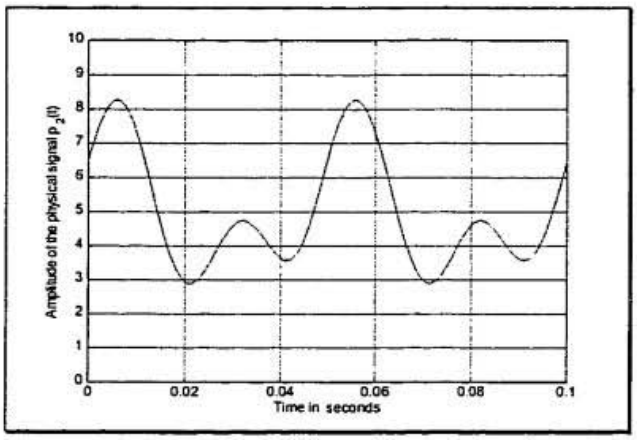


Figure F.5: The second physical signal.

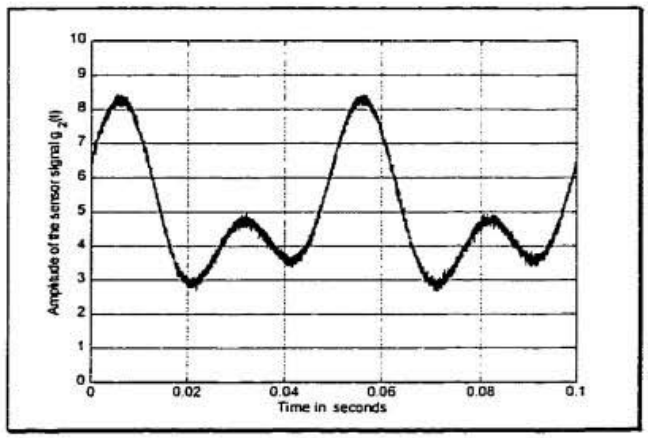


Figure F.6: The second sensor signal with noise.

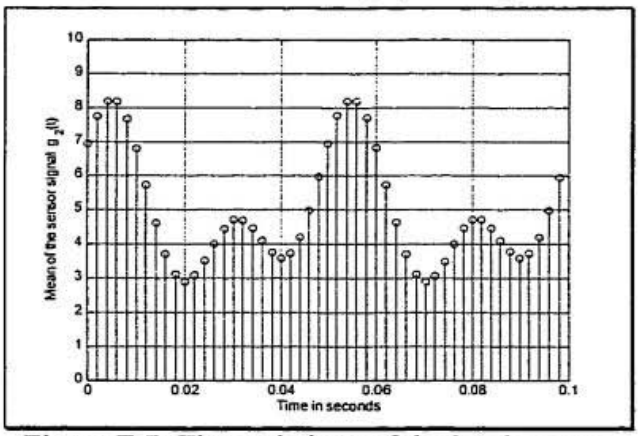


Figure F.7: The variations of the local means.

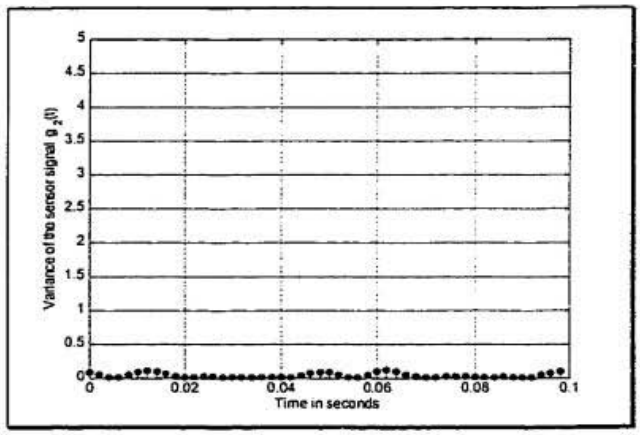


Figure F.8: The variations of local variances.

The Table F.3 shows the major statistics of the physical signal, sensor signal, local means and variances of this sensor signal.

Table F.3: The statistics of the second test signal.

Signals	Maximum	Minimum
$p_2(t)$	8.27	2.90
$g_2(t)$	8.47	2.65
Local means	8.20	2.91
Local variances	0.1169	0.0065

F.2.3 The Characteristics of the Third Test Signal

The formation of the third test signal is given by the following relation

$$f_3(t) = 5 + .5\sin(2\pi 20t) - 1.5\cos(2\pi 20t) + \sin(2\pi 40t) + \cos(2\pi 40t) + 0.5\sin(2\pi 60t) + 0.5\cos(2\pi 60t) \qquad \text{(F.4)}$$

The corresponding physical signal, sensor signal with noise, the local means and variances of the sensor signal are shown in Fig. F.9, Fig.F.10, Fig.F.11, and Fig. F.12 respectively.

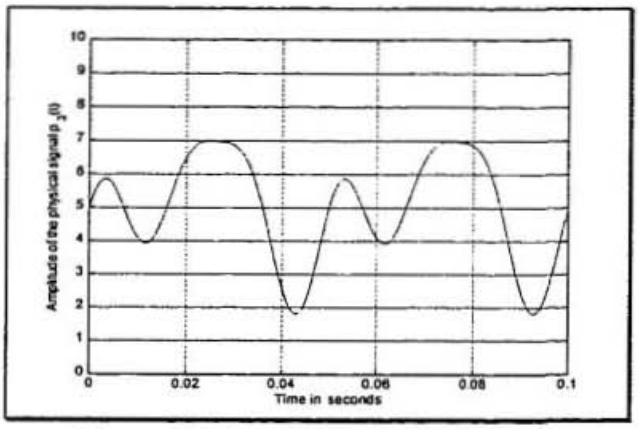


Figure F.9: The third physical signal.

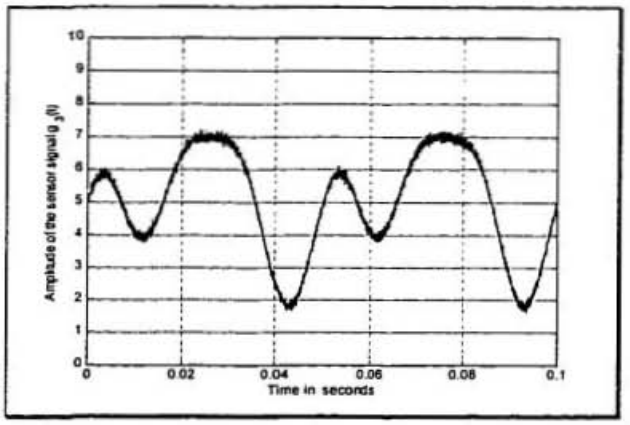


Figure F.10: The third sensor signal with noise.

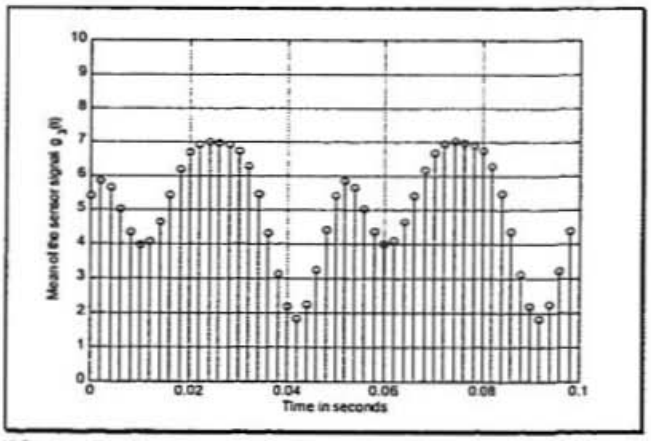


Figure F.11: The local means of the sensor signal.

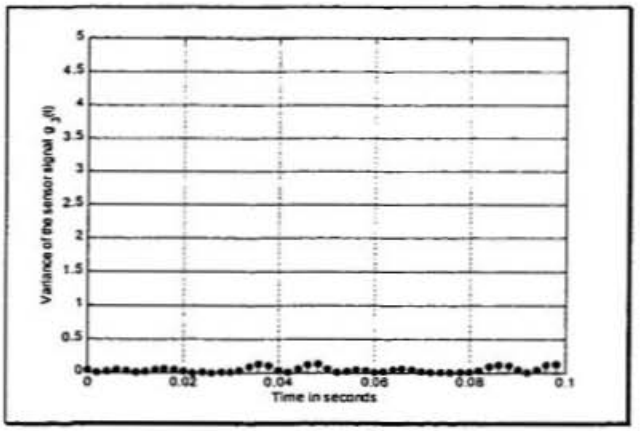


Figure F.12: The local variances of the signal.

The domains of the physical signal, sensor signal, local means and variances are listed in Table F.4.

Table F.4: The statistics of the third signal.

Signals	Maximum	Minimum
$p_3(t)$	7.00	1.80
$g_3(t)$	7.24	1.60
Local means	7.01	1.82
Local variances	0.1298	0.0057

F.2.4 The Characteristics of the Fourth Test Signal

The fourth test signal is formed by the following relation

$$f_3(t) = 5+5\sin(2\pi 60t)+1\cos(2\pi 60t)+0.5\sin(2\pi 120t)+\cos(2\pi 120t)+0.5\sin(2\pi 180t)+0.5\cos(2\pi 180t) \quad \text{(F.5)}$$

The corresponding physical signal, sensor signal with noise, the local means and variances of the sensor signal are shown in Fig. F.13, Fig.F.14, Fig.F.15, and Fig. F.16 respectively.

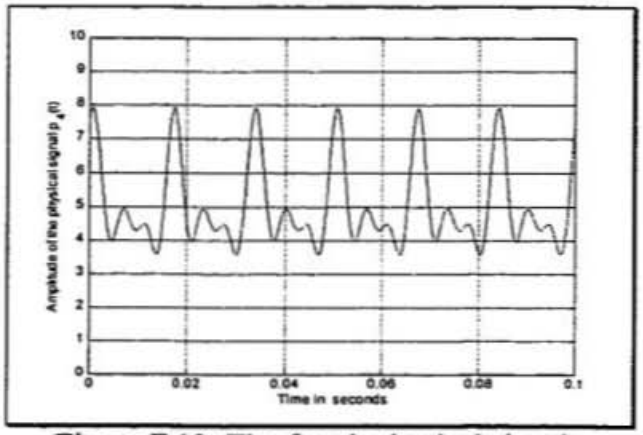


Figure F.13: The fourth physical signal.

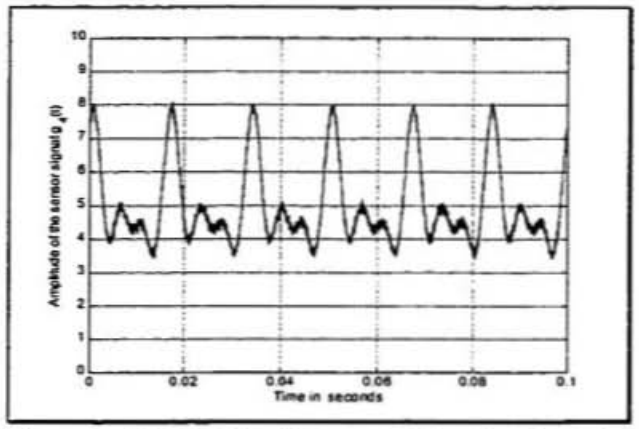


Figure F.14: The fourth sensor signal with noise.

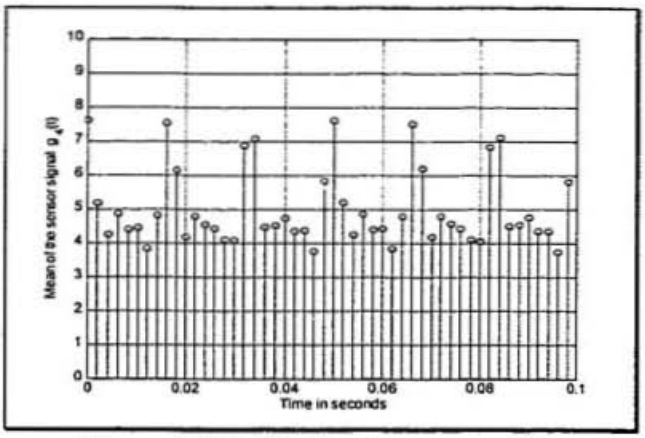


Figure F.15: The local mean profile.

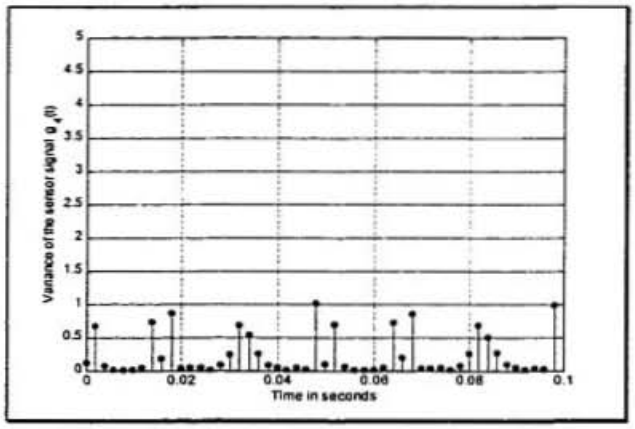


Figure F.16: The local variance profile.

The range of values of the physical signal, sensor signal, local means and variances are listed in Table F.5.

Table F.5: The statistics of the fourth signal.

Signals	Maximum	Minimum
$p_4(t)$	7.918	3.588
$g_4(t)$	8.09	3.40
Local means	7.60	3.72
Local variances	1.006	0.0092

F.3 The Analysis of the Signature of Transient Faults

The following relation has simulated the transient signal as a damped sinusoid:

$$fr(t) = Ae^{-\alpha t} \sin \omega t \tag{F.6}$$

The effect of the transient on the sensor signal has been modeled by superimposition as shown by the following relation.

$$f'(t) = f(t) + f_r(t - t_0)u(t - t_0) \tag{F.7}$$

For the initial analysis the specification of the transient signal is given in Table F.6.

Table F.6: The specification the test transient.

Amplitude, A	Damping coefficient,α	Frequency	Duration
3.0 v	500	1000 Hz	5 ms

This specified transient and a sinusoidal signal corrupted with this transient are shown in Fig. F.17(a) and Fig. F.17(b) respectively.

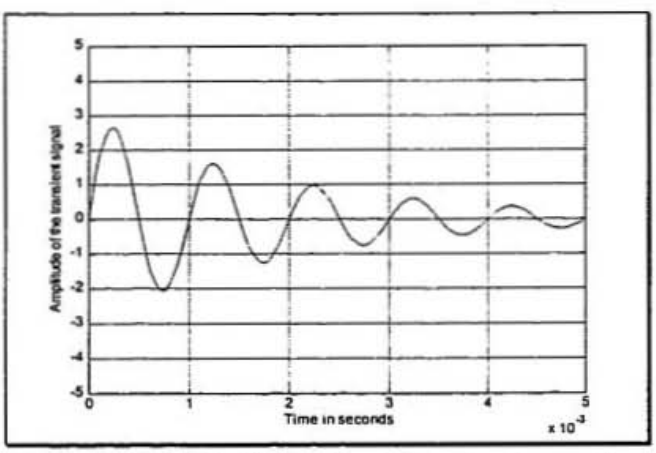


Figure F.17(a): A transient signal as damped sinusoid.

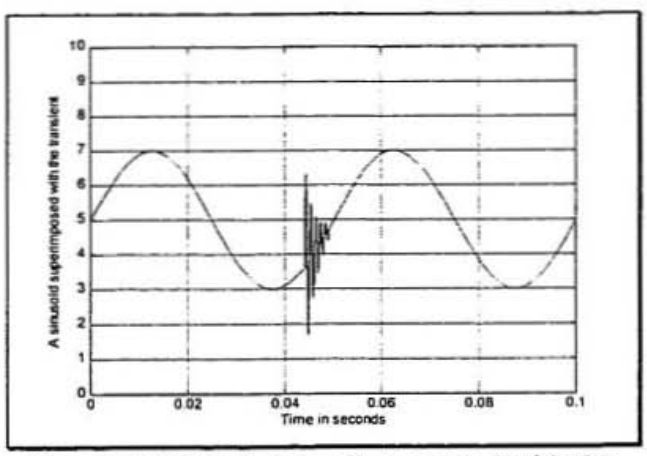


Figure F.17(b): A sinusoid corrupted with the transient.

F.3.1 Signature of the Transient Fault on the First Signal

The local statistics of a 1Khz transient fault of duration 5ms at distance 70ms on the first test signal are shown in Fig. F.19, Fig. F.20, Fig. F.21, and Table-7.

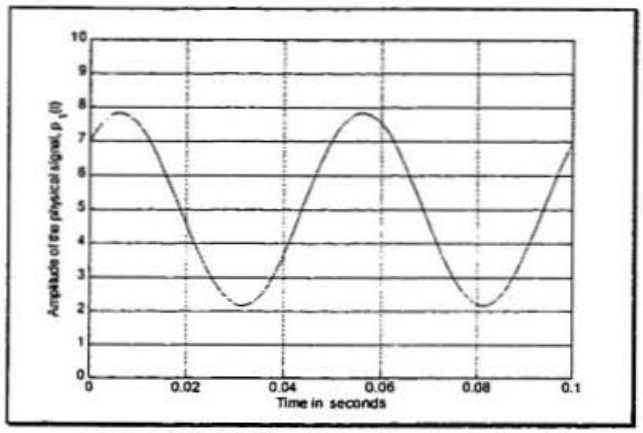


Figure F.18: The first physical signal

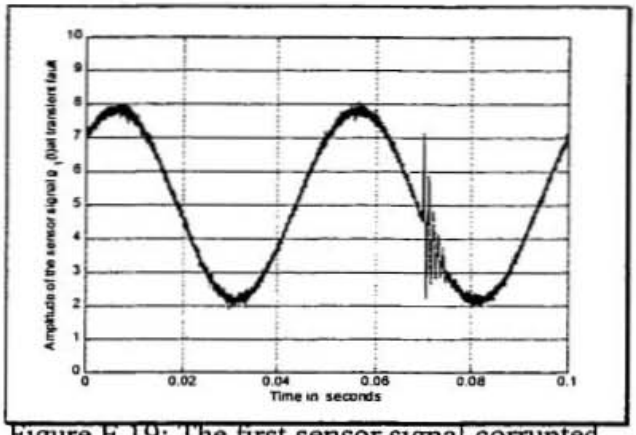


Figure F.19: The first sensor signal corrupted with transient fault.

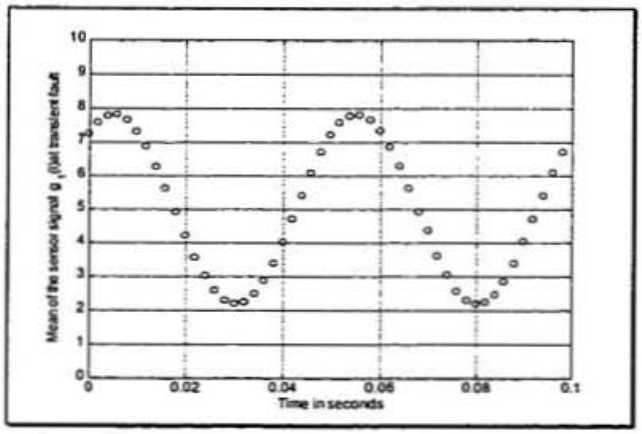


Figure F.20: The local means of the corrupted sensor signal at transient fault.

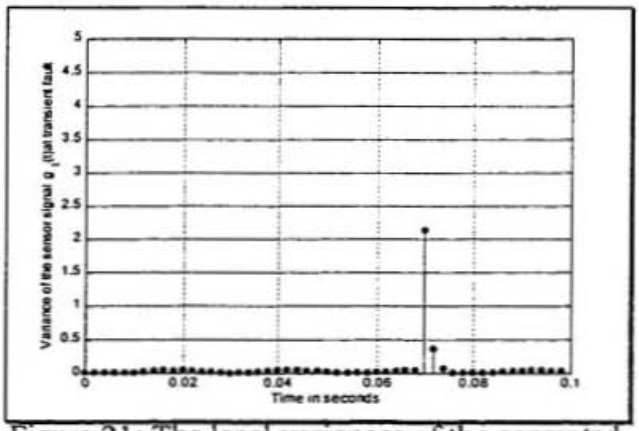


Figure 21: The local variances of the corrupted sensor signal at transient fault.

Table F.7: The statistics of the first test signal at transient fault

Signals	Maximum value		Minimum value	
	Absolute	**Ratio**	Absolute	**Ratio**
$p_1(t)$	7.82	1	2.17	1
$g_1(t)$	8.103	1	1.87	1.01
Local means	7.81	1	2.18	1
Local variance	2.12	**42**	0.005	1

F.3.2 Signature of the Transient Fault on the Second Signal

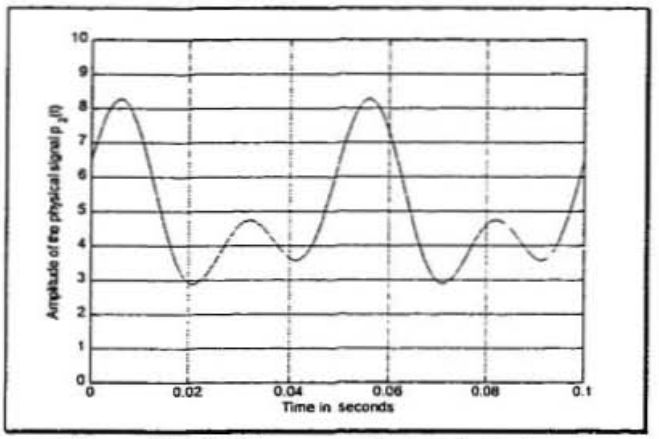


Figure F.22: The second physical signal.

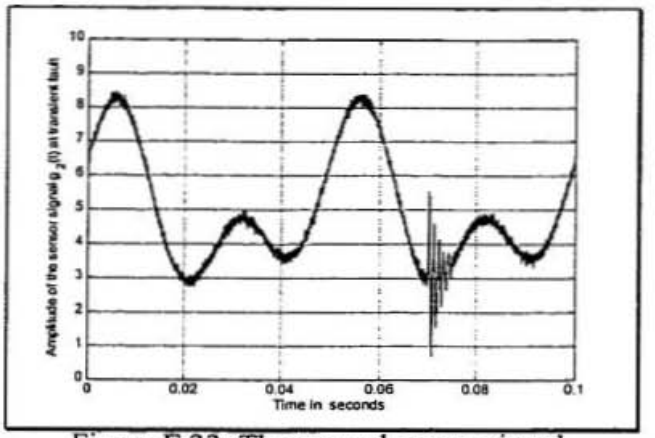


Figure F.23: The second sensor signal superimposed with transient noise.

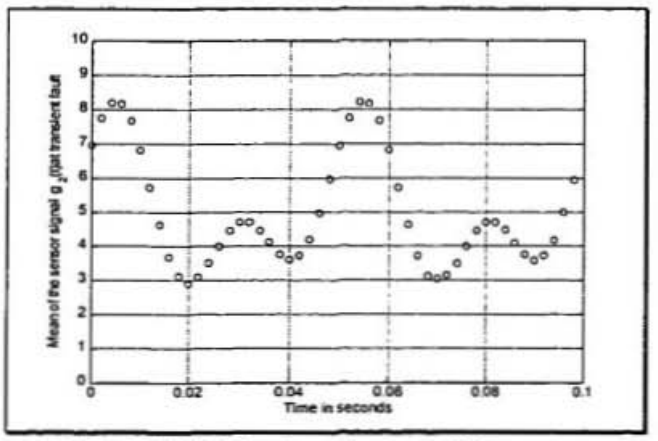


Figure F.24: The local mean of the sensor signal at transient fault.

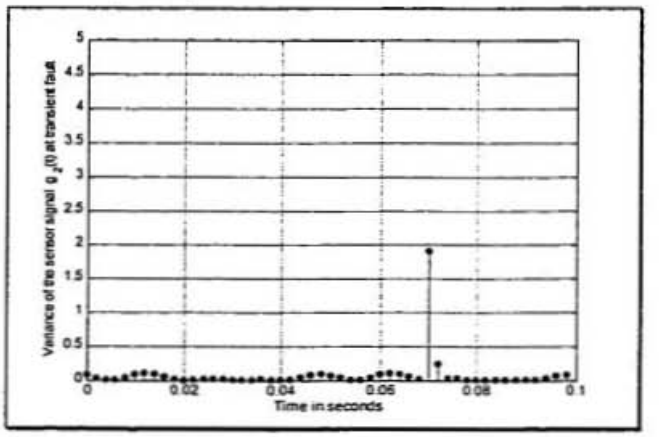


Figure F.25: The local variances of the sensor signal at transient fault.

Table F.8: The statistics of the second test signal at transient fault

Signals	Maximum value		Minimum value	
	Absolute	**Ratio**	Absolute	**Ratio**
$p_2(t)$	8.27	1	2.90	1
$g_2(t)$	8.46	1	.69	0.26
Local means	8.20	1	2.91	1
Local variance	1.9	**16**	.0057	0.87

F.3.3 Signature of the Transient Fault on the Third Test Signal

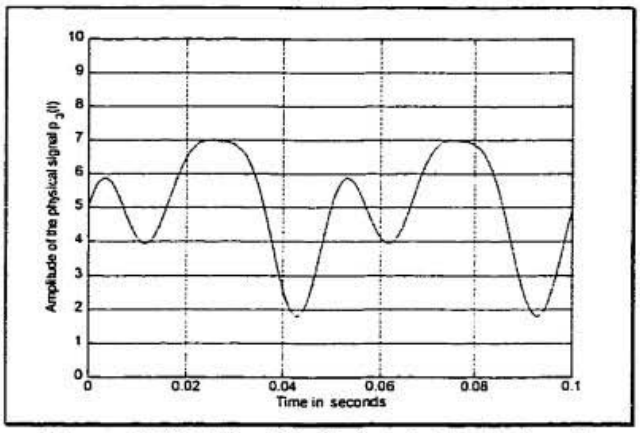


Figure F.26: The third physical test signal.

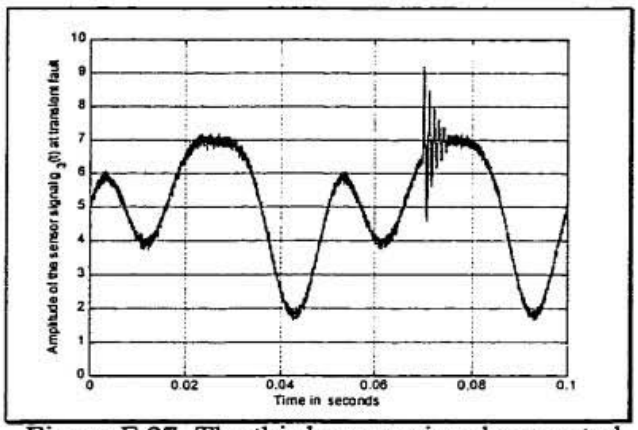


Figure F.27: The third sensor signal corrupted with transient noise.

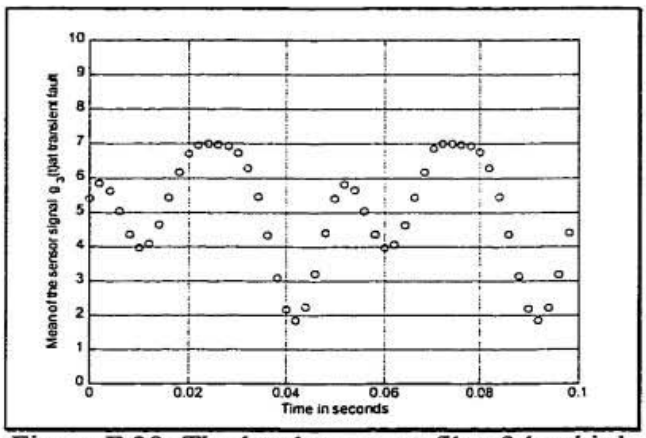


Figure F.28: The local mean profile of the third sensor signal at transient fault.

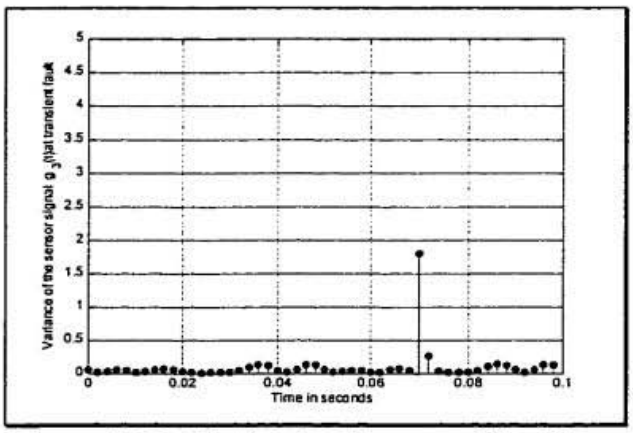


Figure F.29: The local variance profile of the third sensor signal at transient fault.

Table F.9: The statistics of the third test signal at transient fault

Signals	Maximum value		Minimum value	
	Absolute	**Ratio**	Absolute	**Ratio**
$p_3(t)$	7.00	1	1.80	1
$g_3(t)$	9.2	1.27	1.63	1
Local means	7.00	1	1.84	1
Local variance	1.79	**13.79**	0.006	1

F.3.4 Signature of the Transient Fault on the Fourth Test Signal

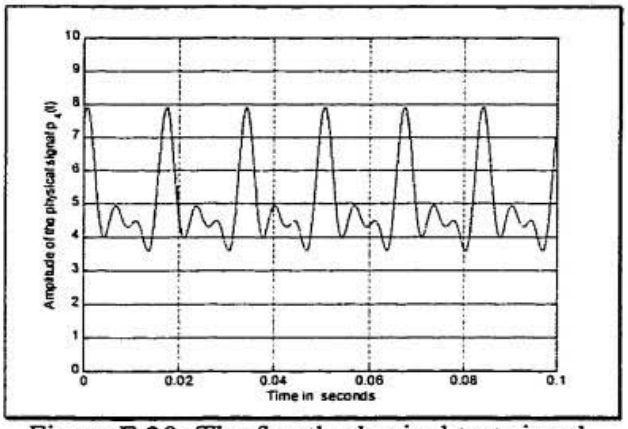


Figure F.30: The fourth physical test signal.

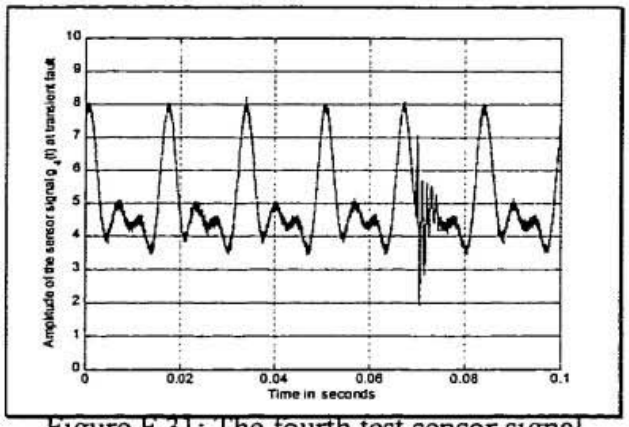


Figure F.31: The fourth test sensor signal corrupted with transient noise.

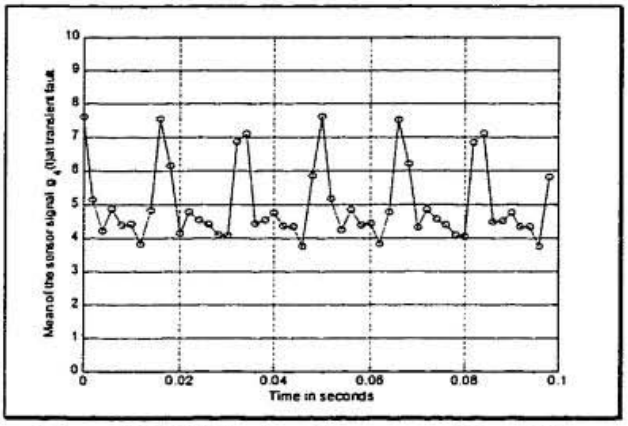


Figure F.32: The local mean profile of the fourth test signal at transient fault.

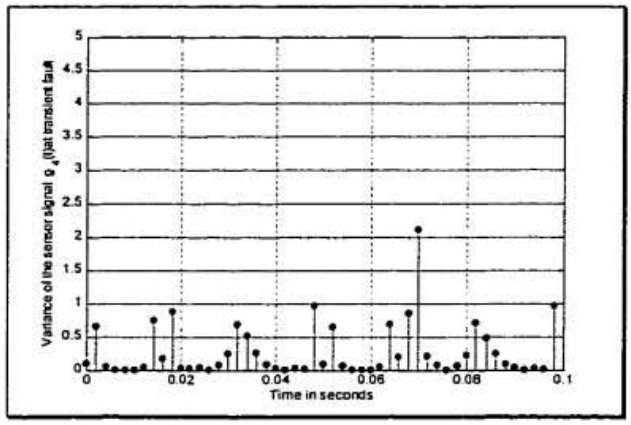


Figure F.33: The local variance profile of the fourth test signal at transient fault.

Table F.10: The statistics of the fourth test signal at transient fault

Signals	Maximum value		Minimum value	
	Absolute	**Ratio**	Absolute	**Ratio**
$p_4(t)$	7.9188	1	3.588	1
$g_4(t)$	8.24	1.01	1.89	0.55
Local means	7.61	1	3.75	1
Local variance	2.122	**2.10**	0.0091	1

F.4.1 The Transient Fault at Different Locations on the First Signal

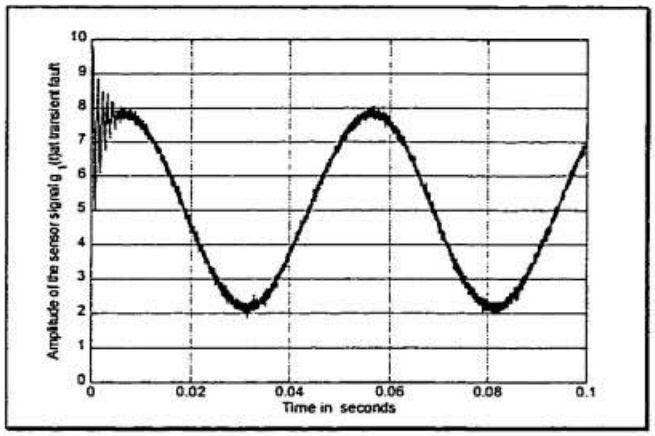


Figure F.34: The transient fault at the origin.

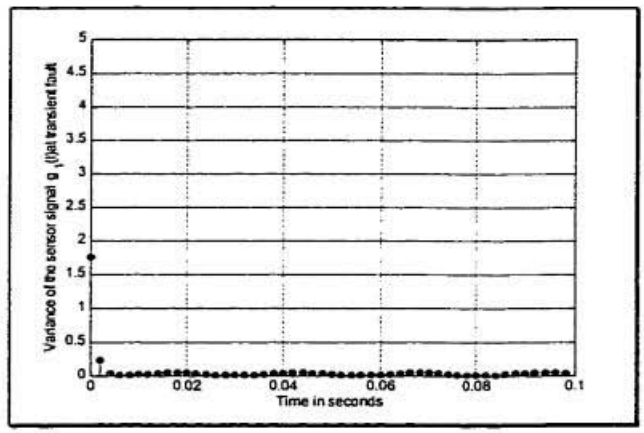


Figure F.35: The variances for the fault at origin.

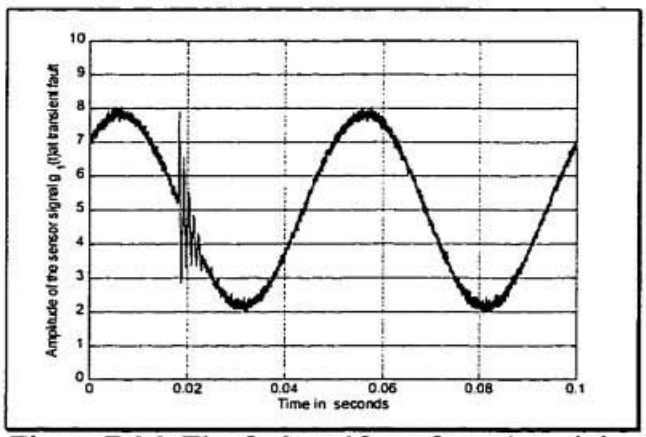


Figure F.36: The fault at 18 ms from the origin.

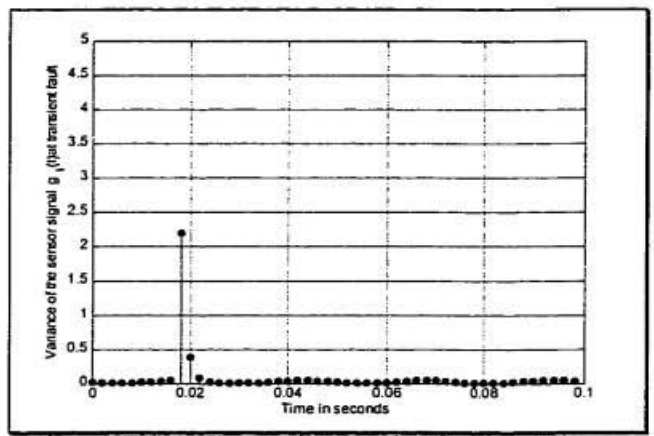


Figure F.37: The variances for fault at 18 ms.

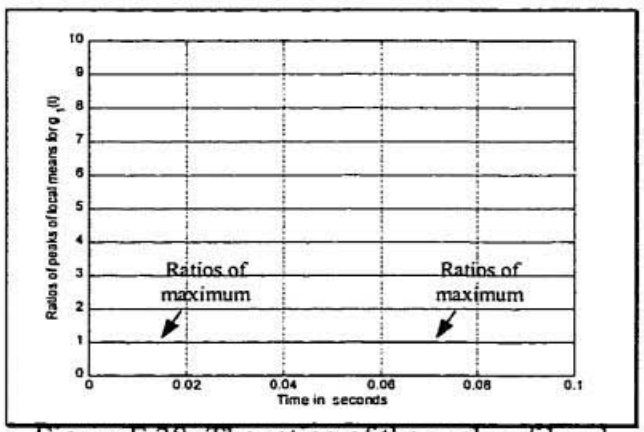


Figure F.38: The ratios of the peaks of local means at transient fault with those at no fault.

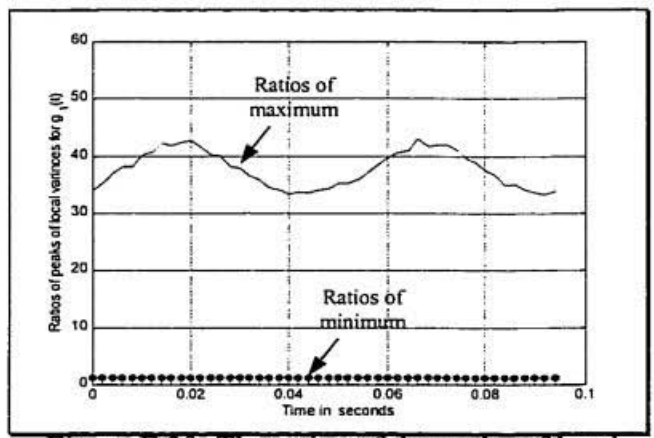


Figure F.39: The ratios of the peaks of local variances at transient fault with those at no fault.

F.4.2 Transient Fault at Various Locations on the Second Signal

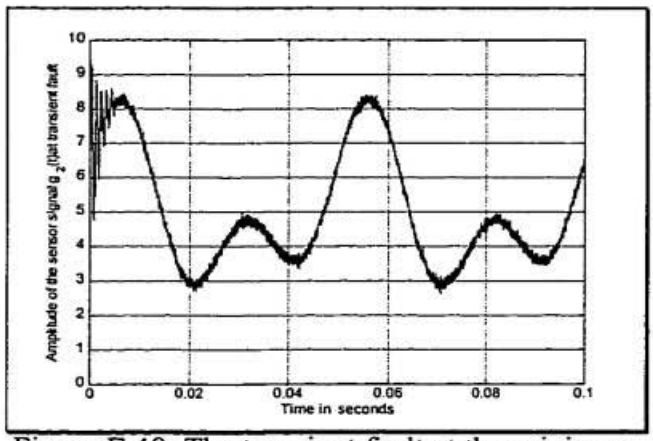


Figure F.40: The transient fault at the origin on the second test signal.

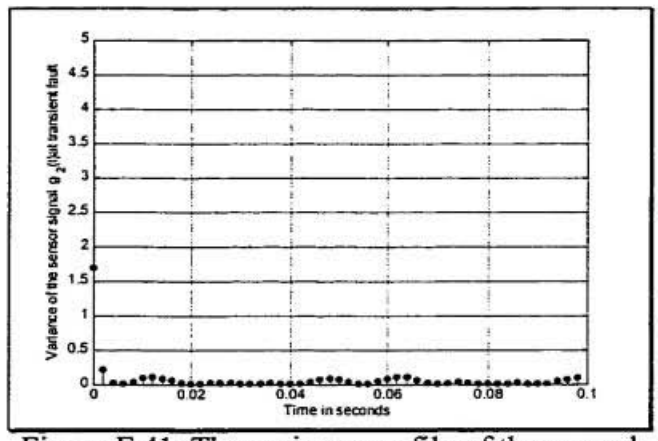


Figure F.41: The variance profile of the second test signal while transient is at the origin.

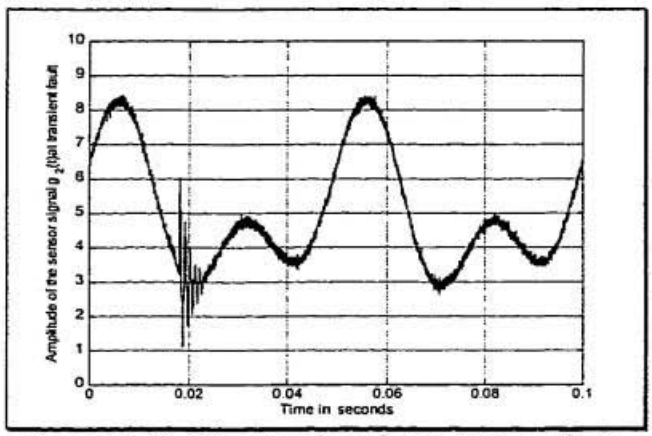


Figure F.42: The transient fault at 18 ms from the origin on second test signal.

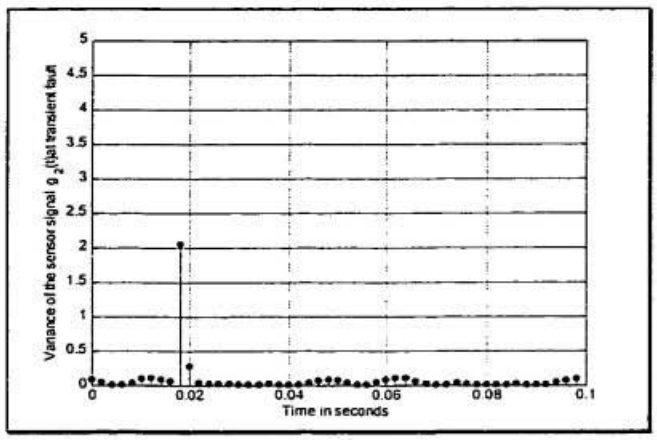


Figure F.43: The variance profile of the second test signal while transient is at 18 ms from the origin.

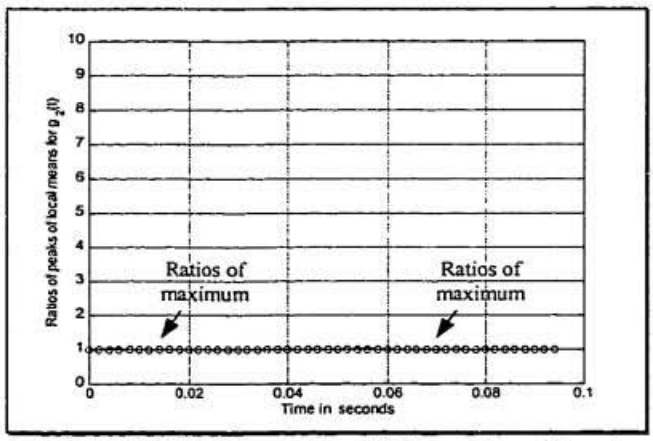


Figure F.44: The ratios of the peaks of local means at transient fault with those at no fault.

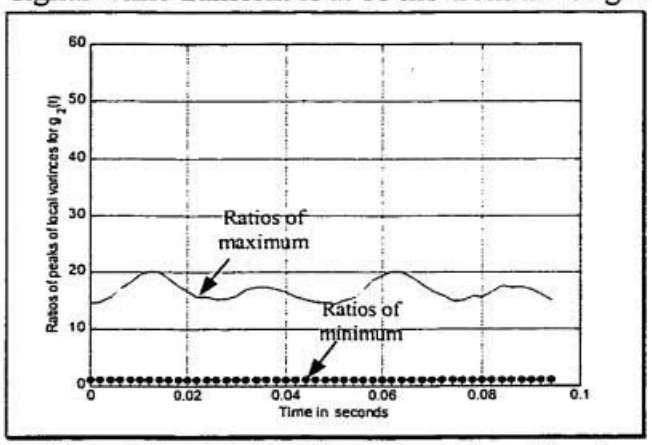


Figure F.45: The ratios of the peaks of local variances at transient fault with those at no fault.

F.4.3 Transient Fault at Various Locations on the Third Signal

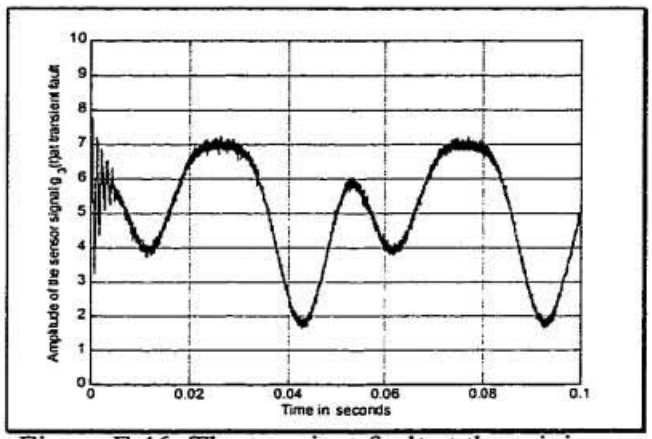


Figure F.46: The transient fault at the origin on the third test signal.

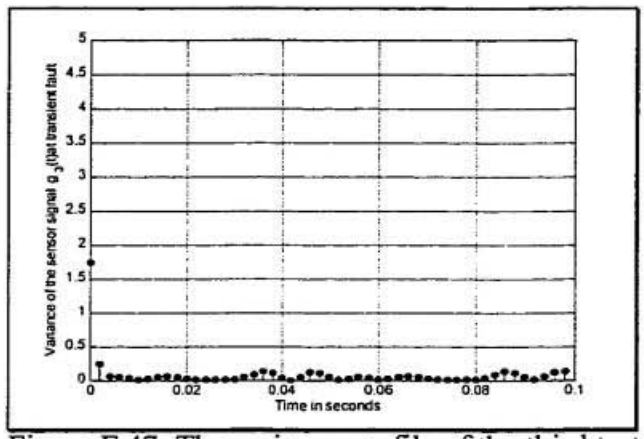


Figure F.47: The variance profile of the third test signal while transient is at the origin.

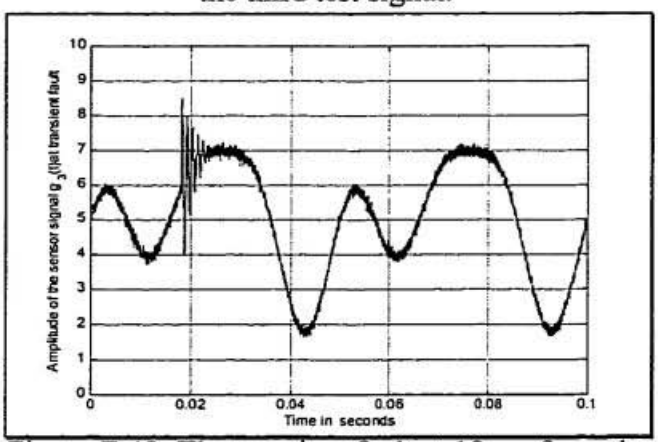


Figure F.48: The transient fault at 18 ms from the origin on third test signal.

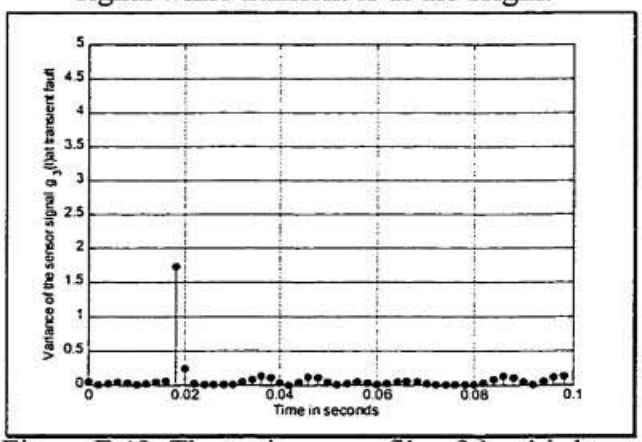


Figure F.49: The variance profile of the third test signal while transient is at 18 ms from the origin.

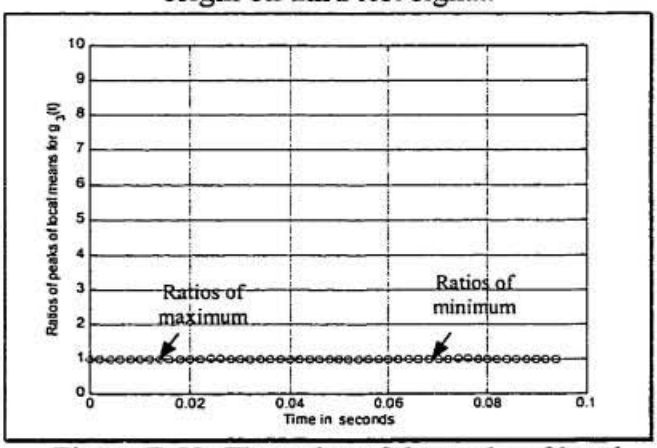


Figure F.50: The ratios of the peaks of local means at transient fault with those at no fault.

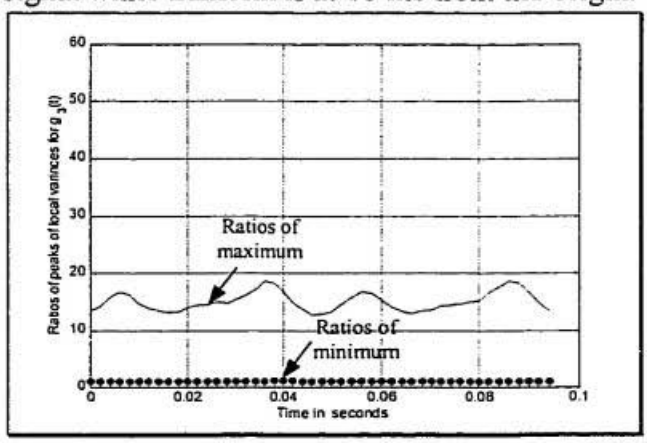


Figure F.51: The ratios of the peaks of local variances at transient fault with those at no fault.

F.4.4 Transient Fault at Various Locations on the Fourth Signal

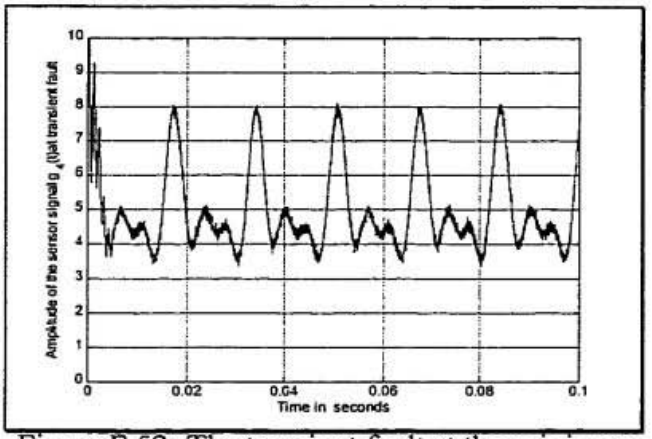


Figure F.52: The transient fault at the origin on the fourth test signal.

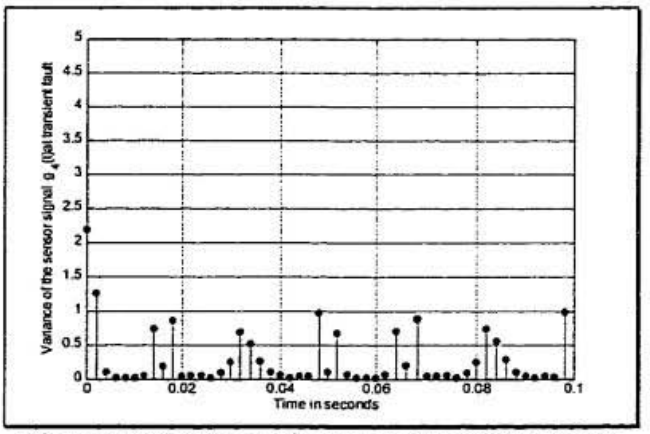


Figure F.53: The variance profile of the fourth test signal while transient is at the origin.

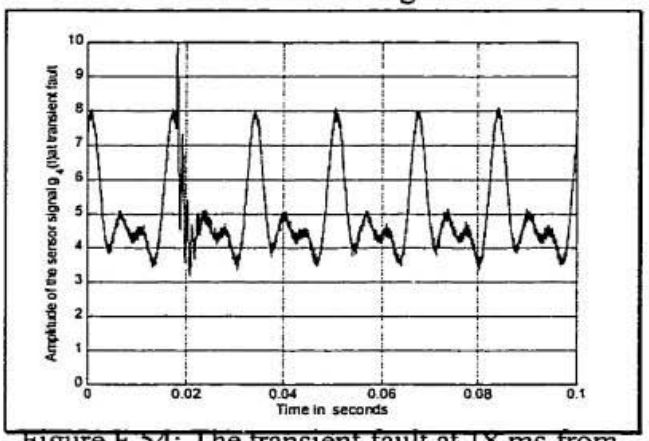


Figure F.54: The transient fault at 18 ms from the origin on the fourth test signal.

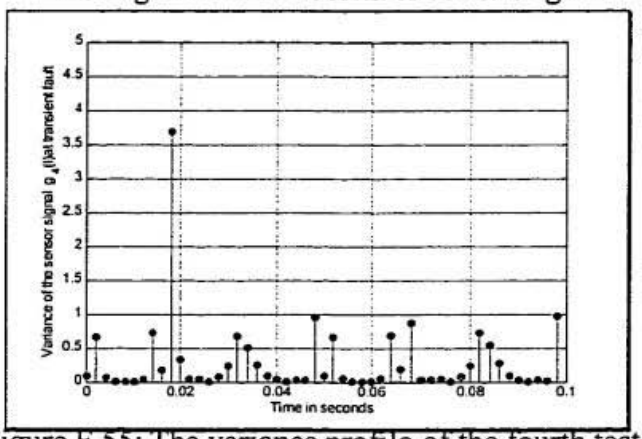


Figure F.55: The variance profile of the fourth test signal while transient is at 18 ms from the origin.

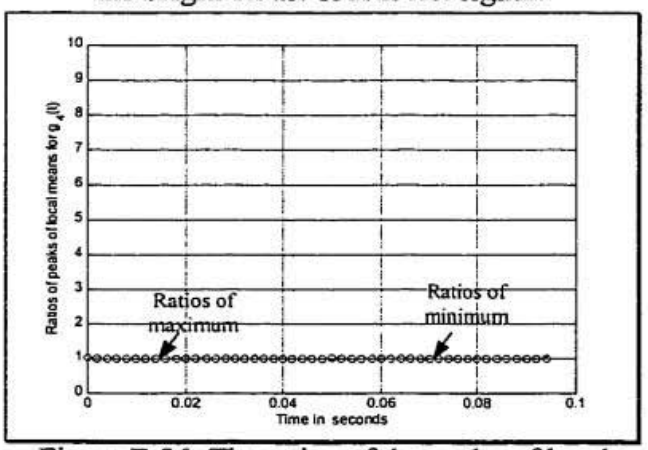


Figure F.56: The ratios of the peaks of local means at transient fault with those at no fault.

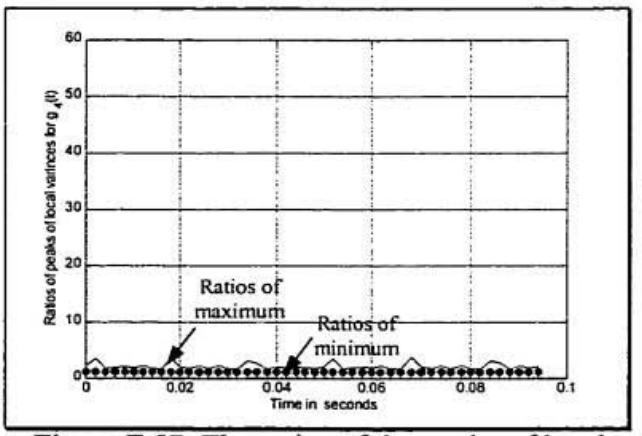


Figure F.57: The ratios of the peaks of local variances at transient fault with those at no fault.

F.5.1 The Effect of Window Size on Local Statistics at Transient Fault on the First Test Signal

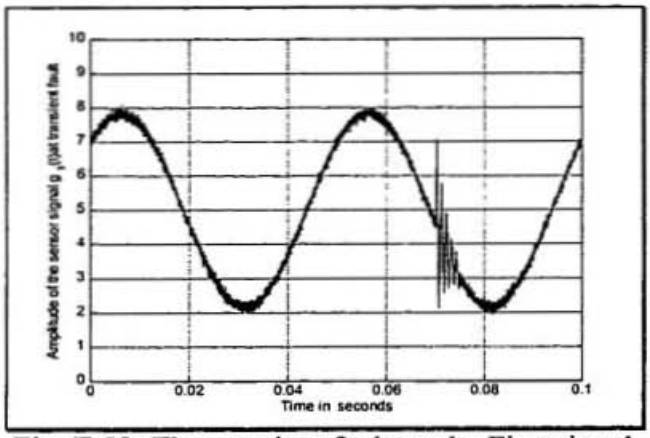


Fig. F.58: The transient fault on the First signal.

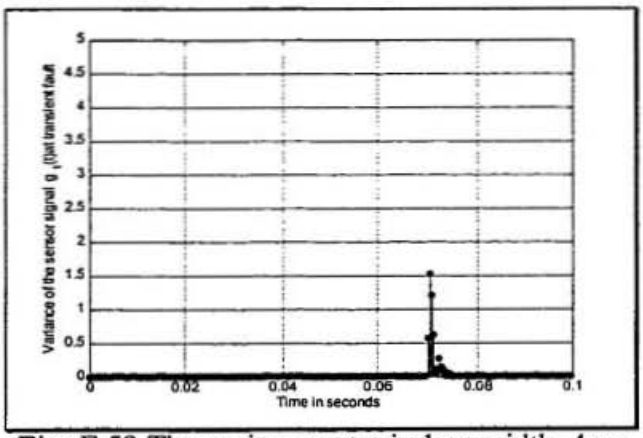


Fig. F.59:The variances at window width .4ms.

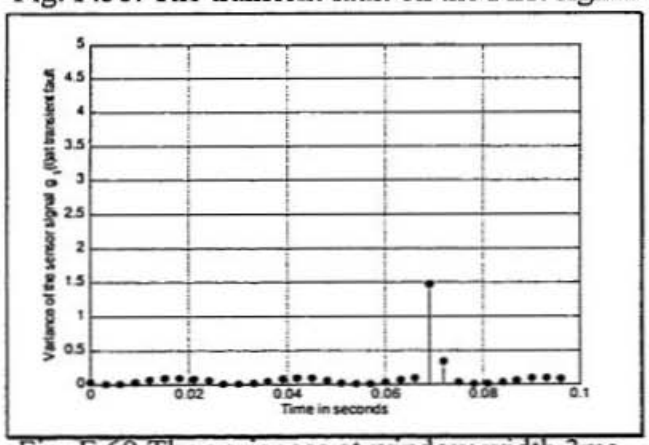


Fig. F.60:The variances at window width 3ms.

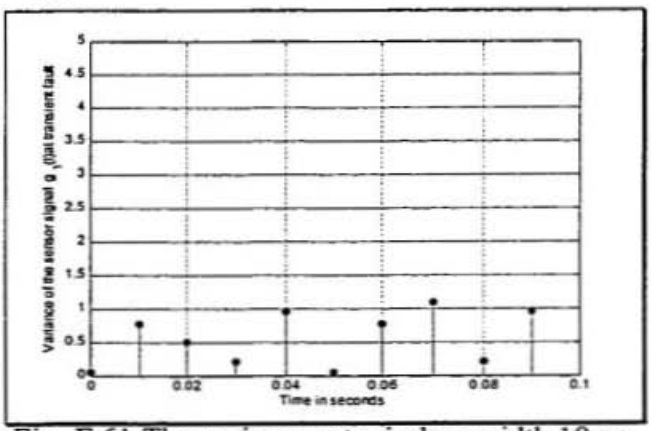


Fig. F.61:The variances at window width 10ms.

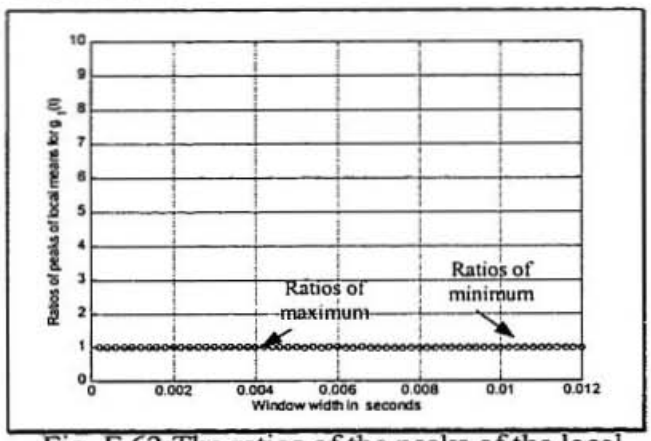


Fig. F.62:The ratios of the peaks of the local means at different window widths.

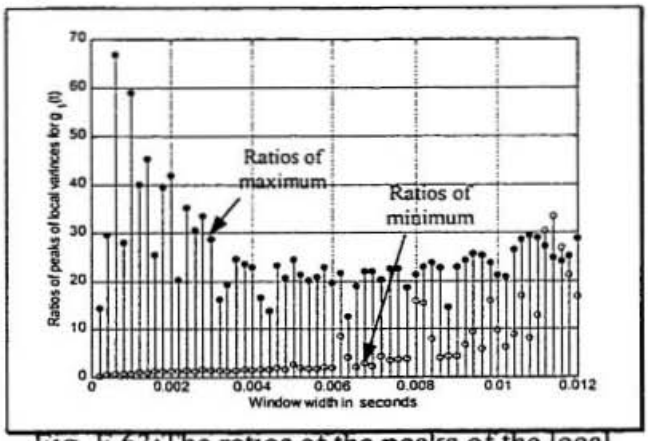


Fig. F.63:The ratios of the peaks of the local variances at different window widths.

F.5.2 The Effect of Window Size on Local Statistics at Transient Fault on the Second Test Signal

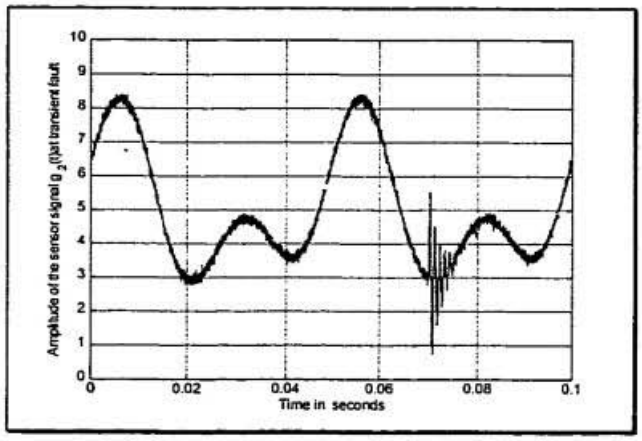


Fig. F.64: The transient fault on the 2nd signal.

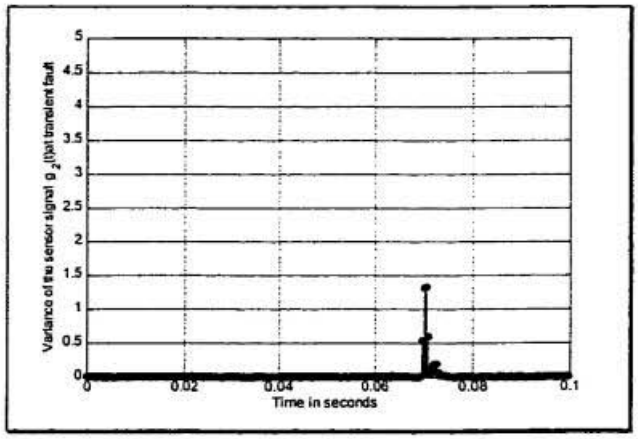


Fig. F.65:The variances at window width .4ms.

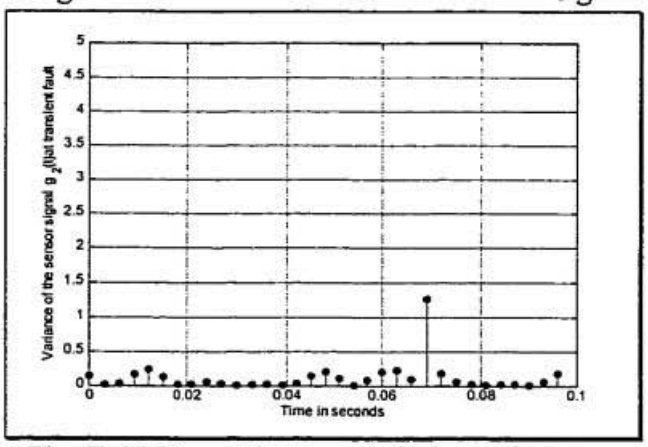


Fig. F.66:The variances at window width 3ms.

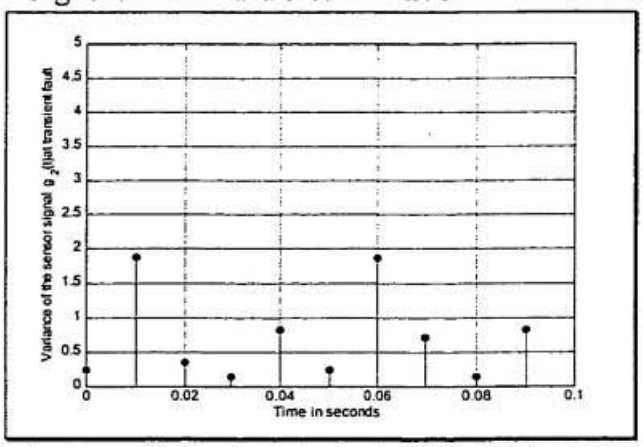


Fig. F.67:The variances at window width 10ms.

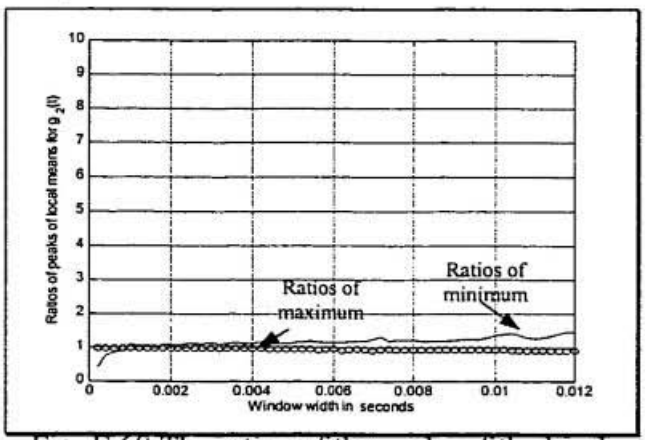


Fig. F.68:The ratios of the peaks of the local means at different window widths.

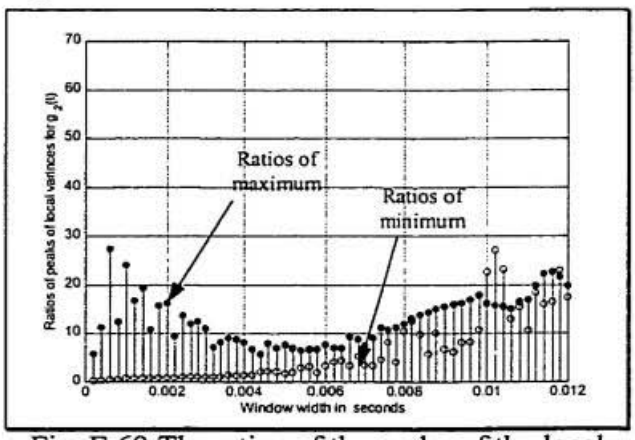


Fig. F.69:The ratios of the peaks of the local variances at different window widths.

F.5.3 The Effect of Window Size on Local Statistics at Transient Fault on the Third Test Signal

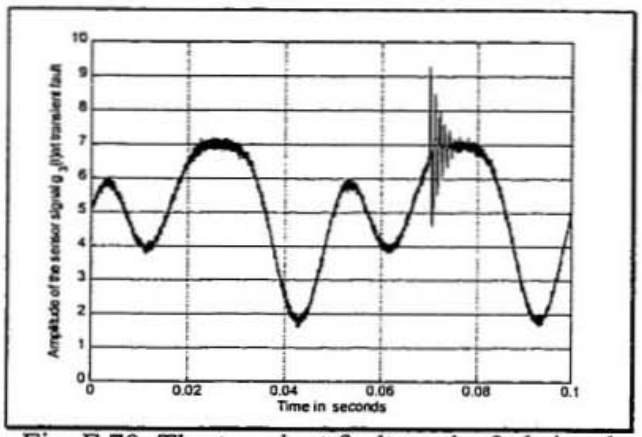


Fig. F.70: The transient fault on the 3rd signal.

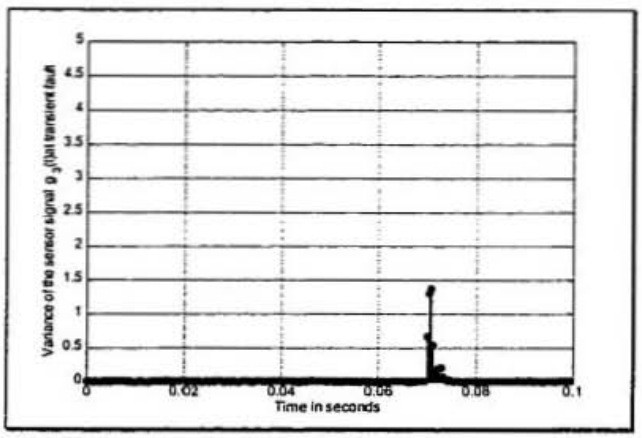


Fig. F.71:The variances at window width .4ms.

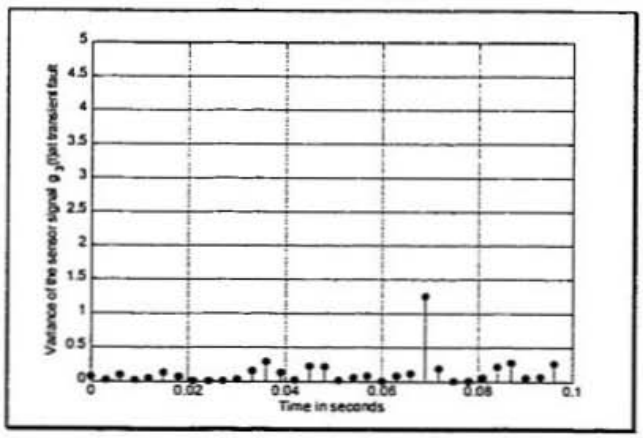


Fig. F.72:The variances at window width 3ms.

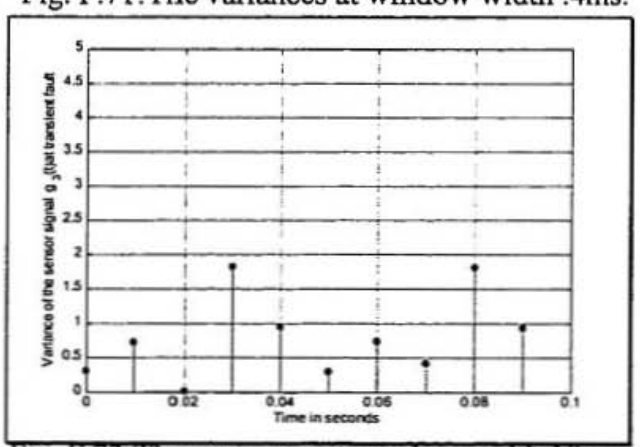


Fig. F.73:The variances at window width 10ms.

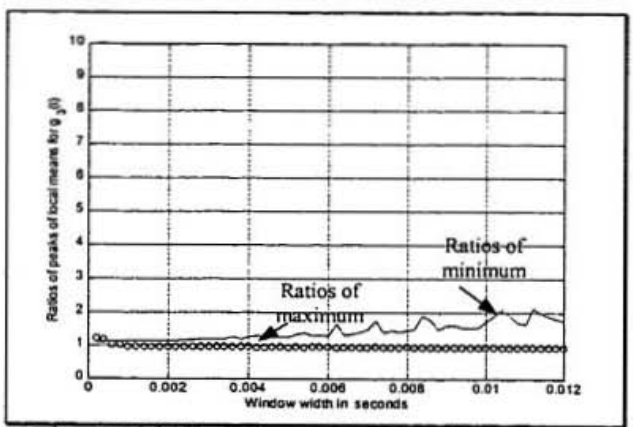


Fig. F.74:The ratios of the peaks of the local means at different window widths.

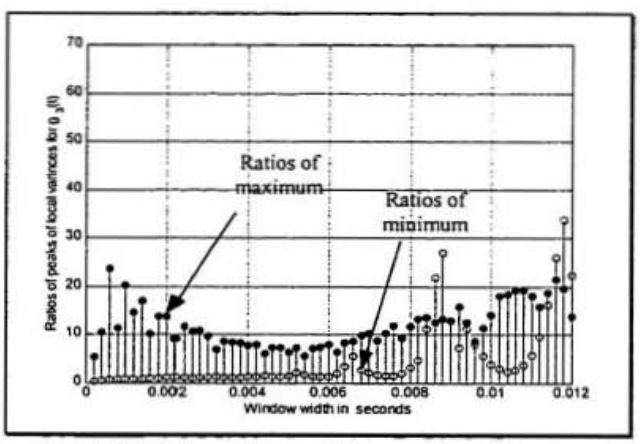


Fig. F.75:The ratios of the peaks of the local variances at different window widths.

F.5.4 The Effect of Window Size on Local Statistics at Transient Fault on the Fourth Test Signal

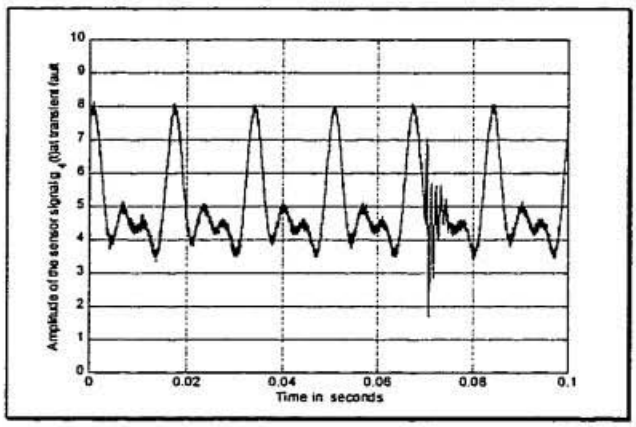


Fig. F.76: The transient fault on the 4th signal.

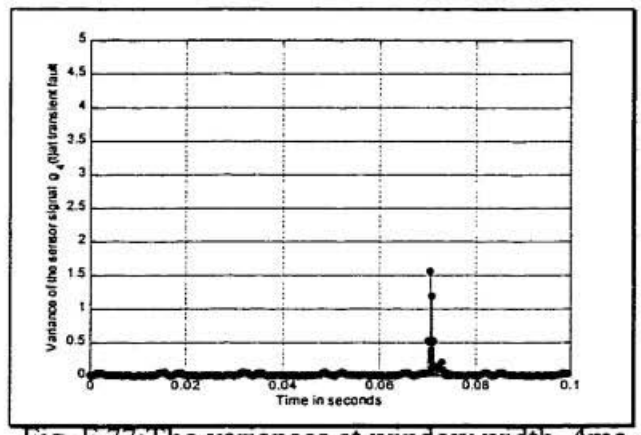


Fig. F.77:The variances at window width .4ms.

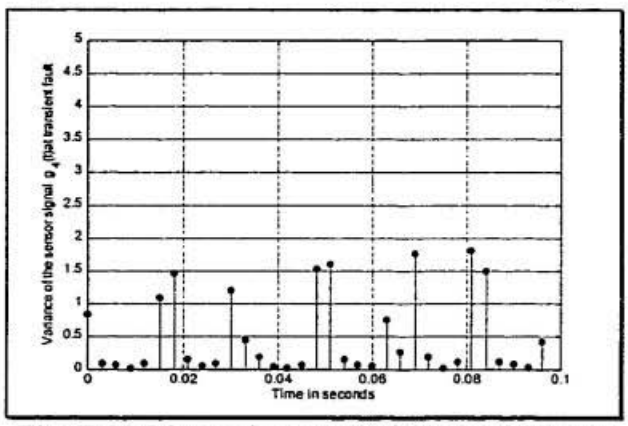


Fig. F.78:The variances at window width 3ms.

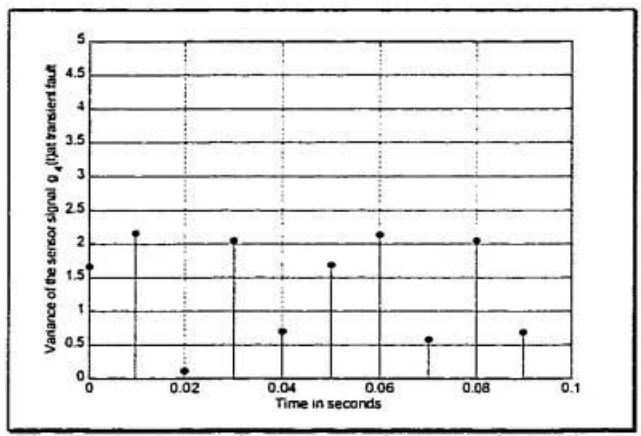


Fig. F.79:The variances at window width 10ms.

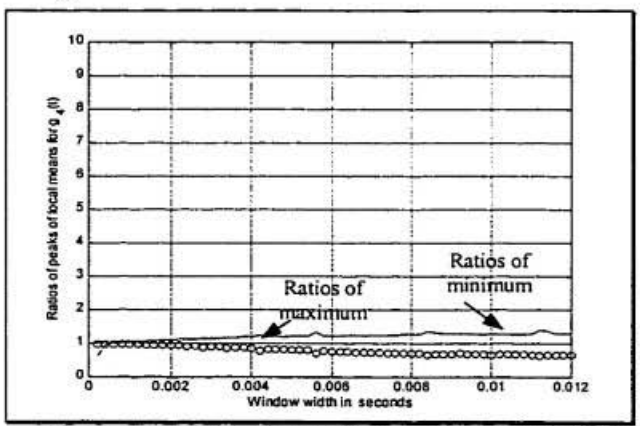


Fig. F.80:The ratios of the peaks of the local means at different window widths.

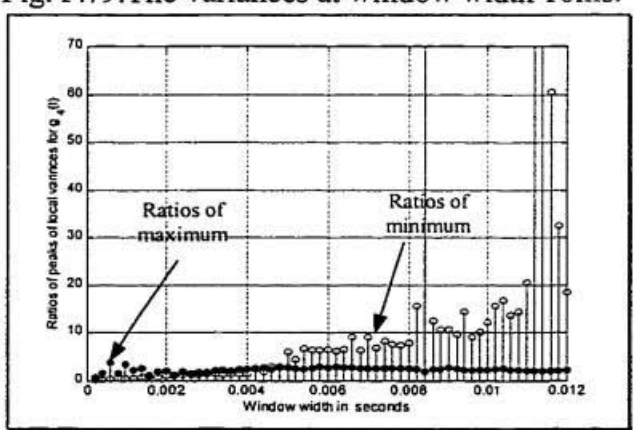


Fig. F.81:The ratios of the peaks of the local variances at different window widths.

F.6.1 The Effect of Window Locations on Local Statistics at Transient Fault on the First Test Signal

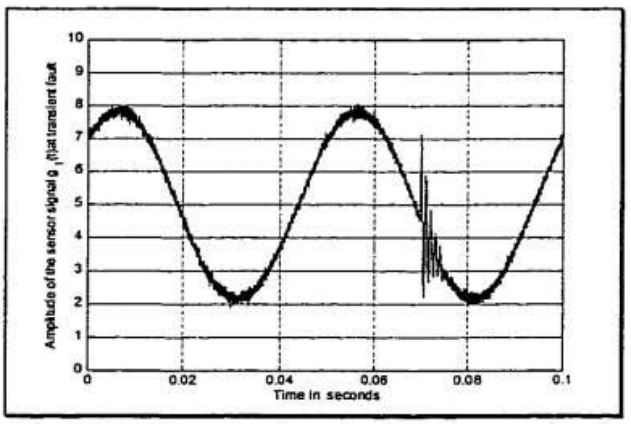


Fig. F.82: The transient fault on the 1st signal.

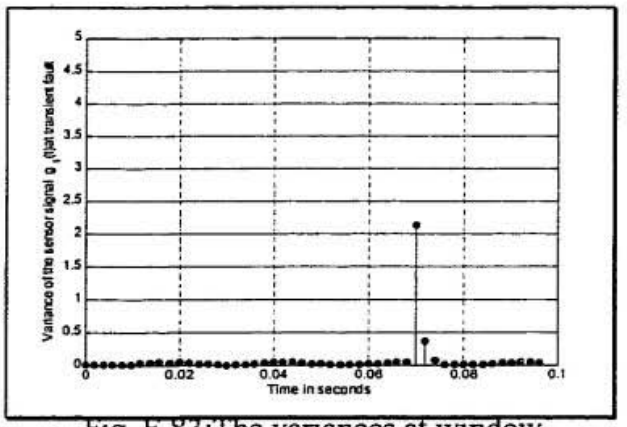


Fig. F.83:The variances at window displacement of .02 ms from the origin.

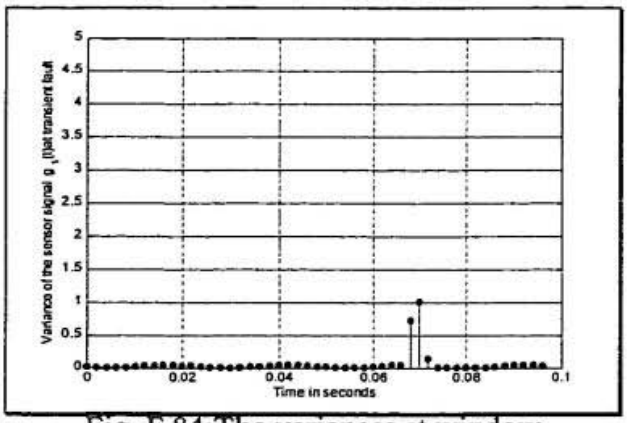


Fig. F.84:The variances at window displacement of .66 ms from the origin.

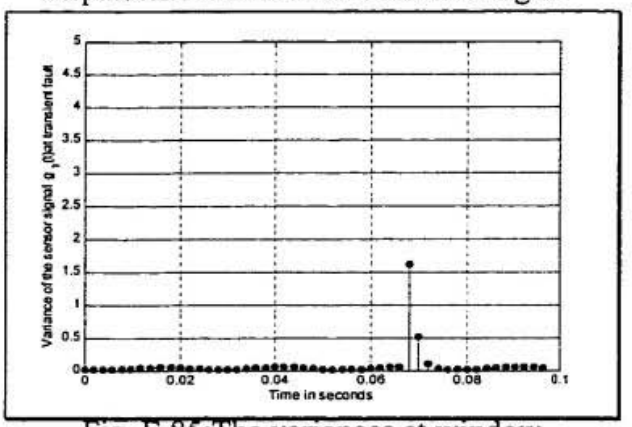


Fig. F.85:The variances at window displacement of 1.2 ms from the origin.

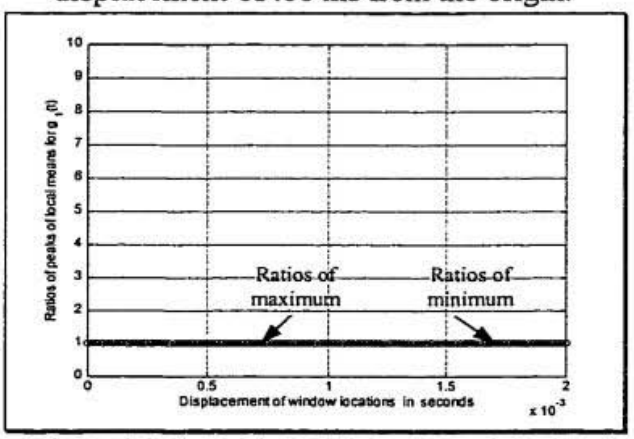


Fig. F.86:The ratios of the peaks of the local means at different window displacements.

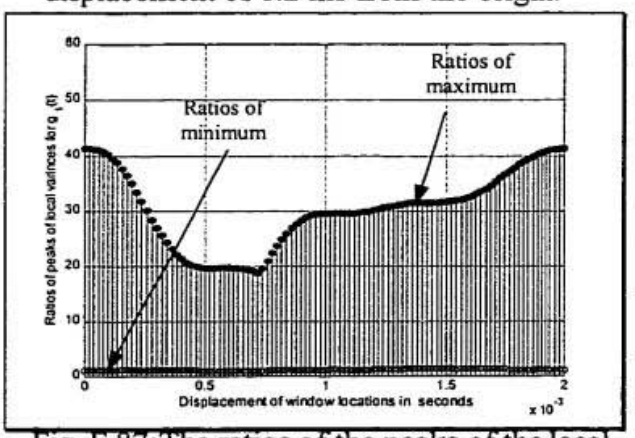


Fig. F.87:The ratios of the peaks of the local variances at different window displacements.

F.6.2 The Effect of Window Locations on Local Statistics at Transient Fault on the Second Test Signal

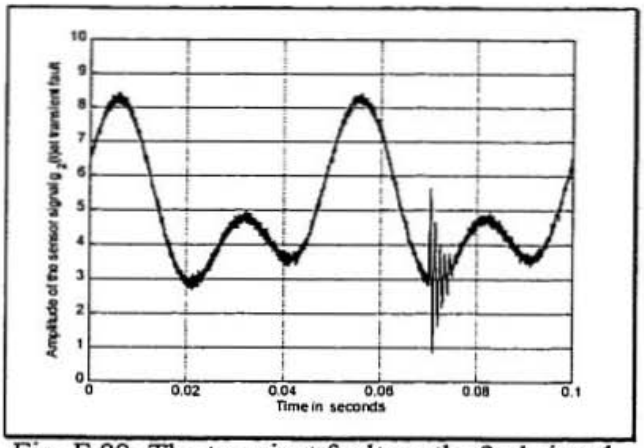


Fig. F.88: The transient fault on the 2nd signal.

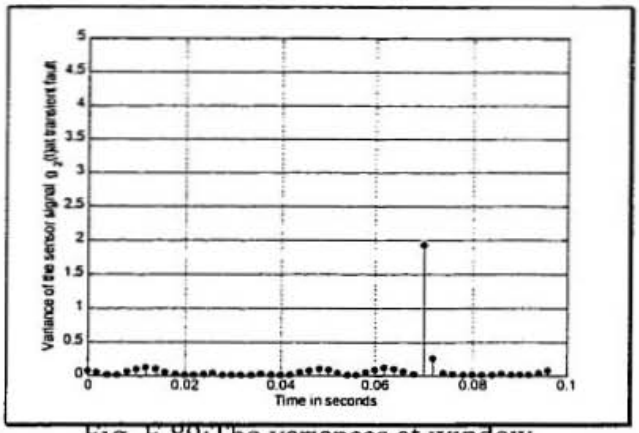


Fig. F.89:The variances at window displacement of .02 ms from the origin.

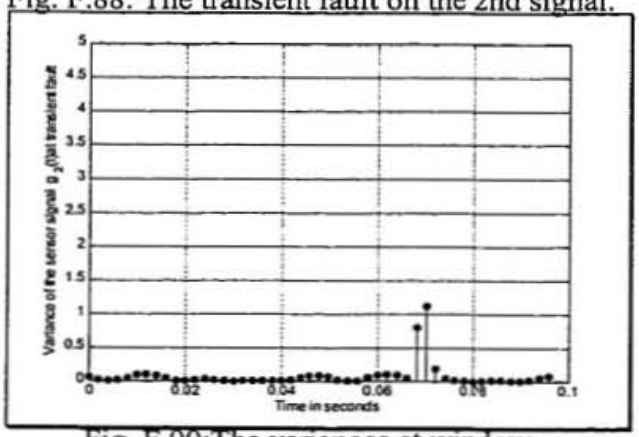


Fig. F.90:The variances at window displacement of .66 ms from the origin.

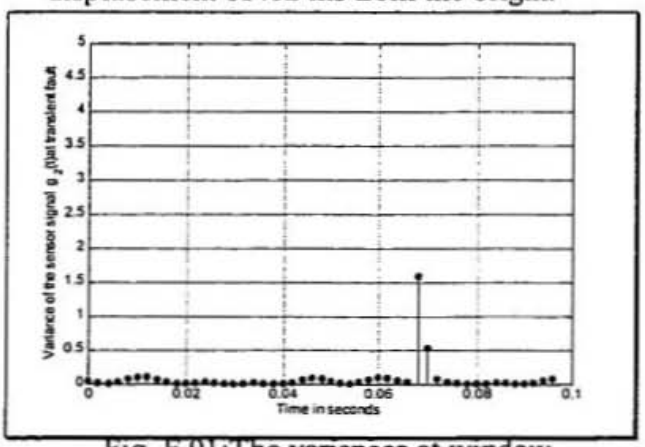


Fig. F.91:The variances at window displacement of 1.2 ms from the origin.

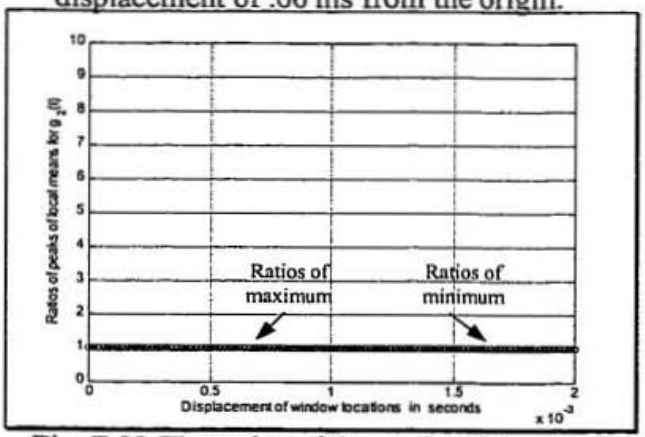


Fig. F.92:The ratios of the peaks of the local means at different window displacements.

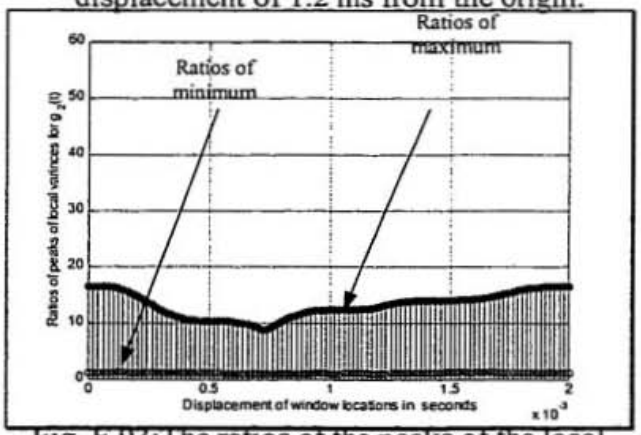


Fig. F.93:The ratios of the peaks of the local variances at different window displacements.

F.6.3 The Effect of Window Locations on Local Statistics at Transient Fault on the Third Test Signal

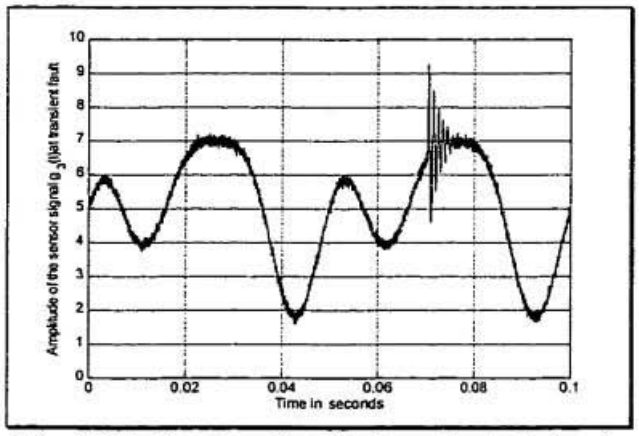


Fig. F.94: The transient fault on the 3rd signal.

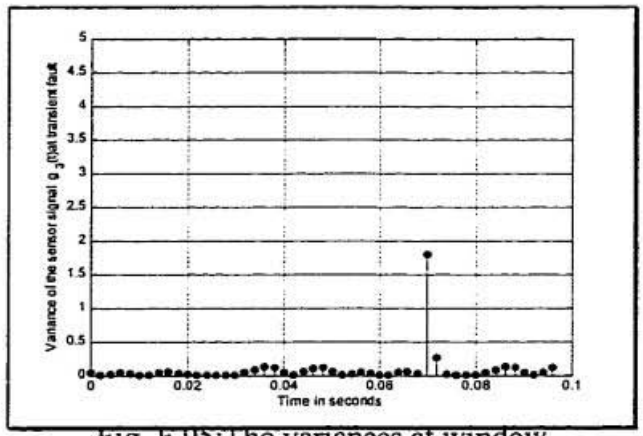


Fig. F.95:The variances at window displacement of .02 ms from the origin.

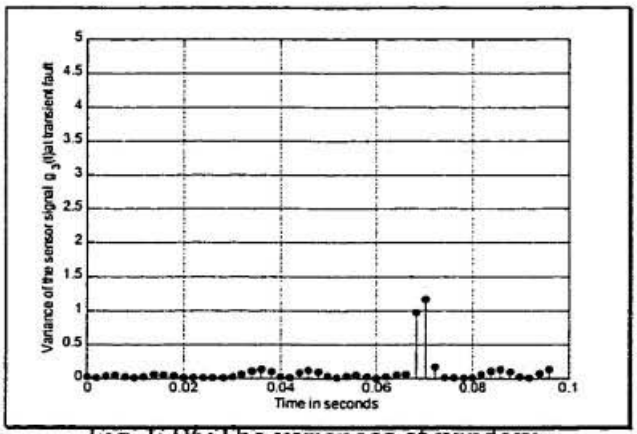


Fig. F.96:The variances at window displacement of .66 ms from the origin.

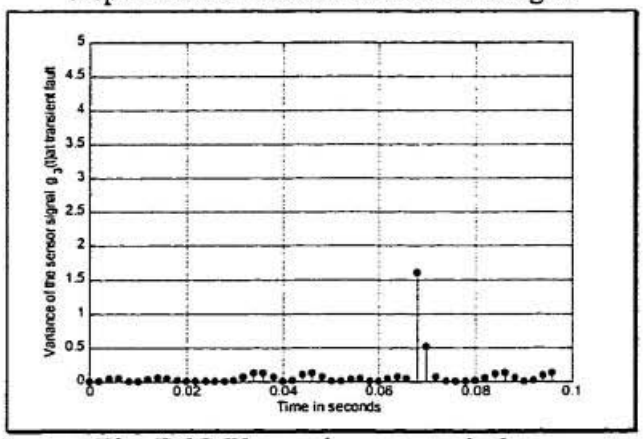


Fig. F.97:The variances at window displacement of 1.2 ms from the origin.

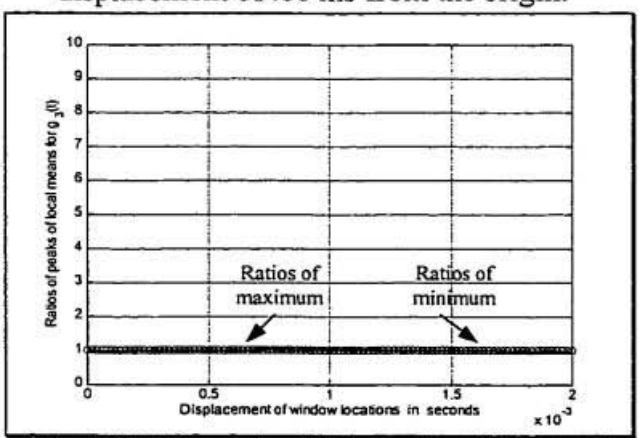


Fig. F.98:The ratios of the peaks of the local means at different window displacements.

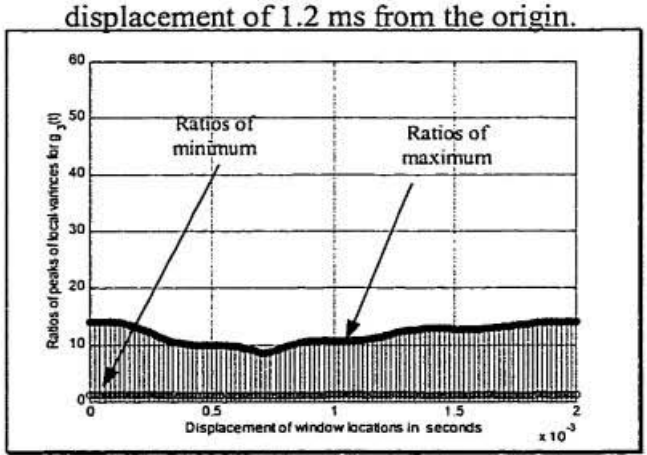


Fig. F.99:The ratios of the peaks of the local variances at different window displacements.

F.6.4 The Effect of Window Locations on Local Statistics at Transient Fault on the Fourth Test Signal

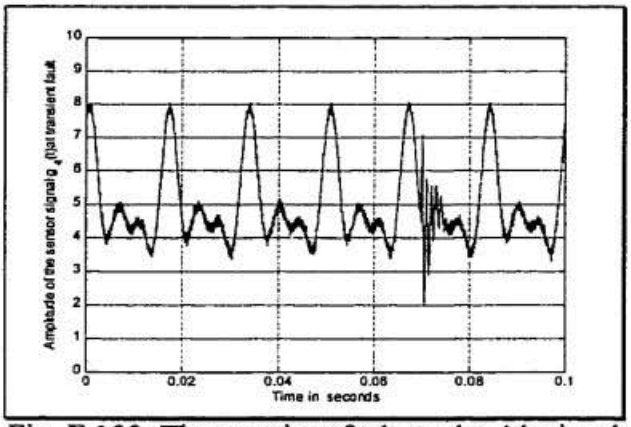


Fig. F.100: The transient fault on the 4th signal.

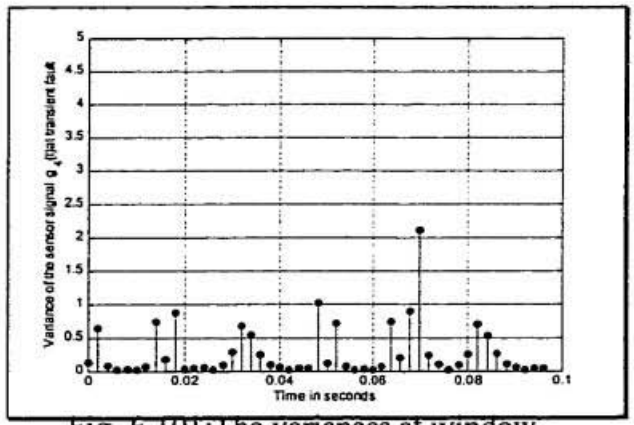


Fig. F.101:The variances at window displacement of .02 ms from the origin.

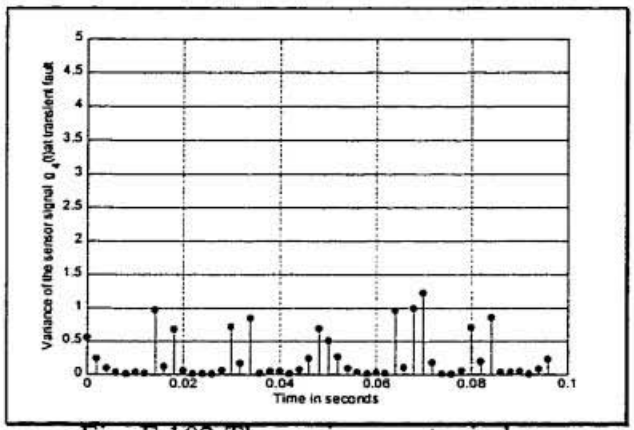


Fig. F.102:The variances at window displacement of .66 ms from the origin.

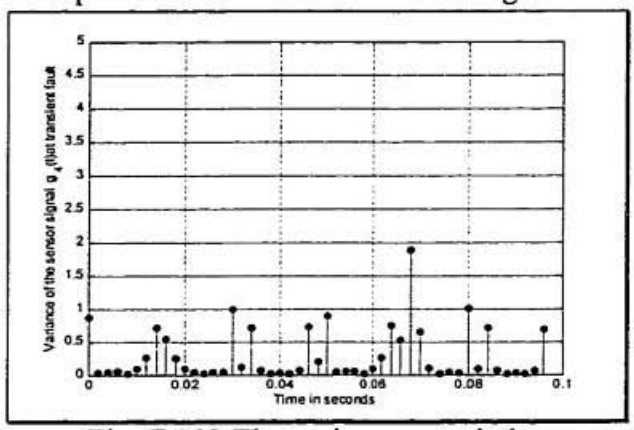


Fig. F.103:The variances at window displacement of 1.2 ms from the origin.

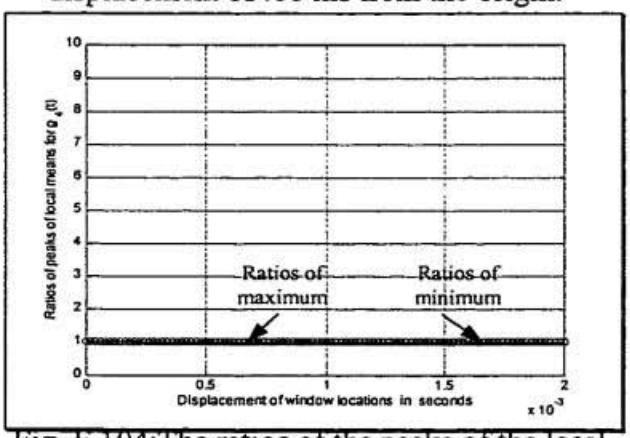


Fig. F.104:The ratios of the peaks of the local means at different window displacements.

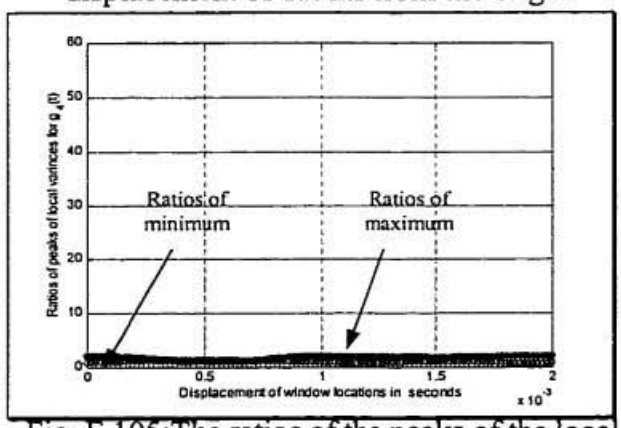


Fig. F.105:The ratios of the peaks of the local variances at different window displacements.

F.7.1 The Effect of Transient Faults of Different Frequencies on the Local Statistics of the First Test Signal

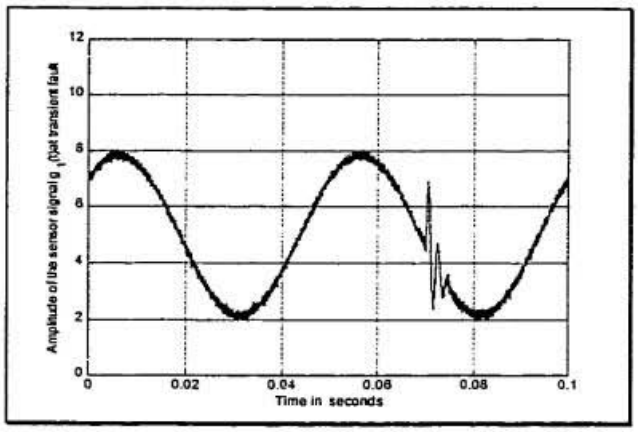


Fig. F.106: The 500 Hz transient on 1st signal.

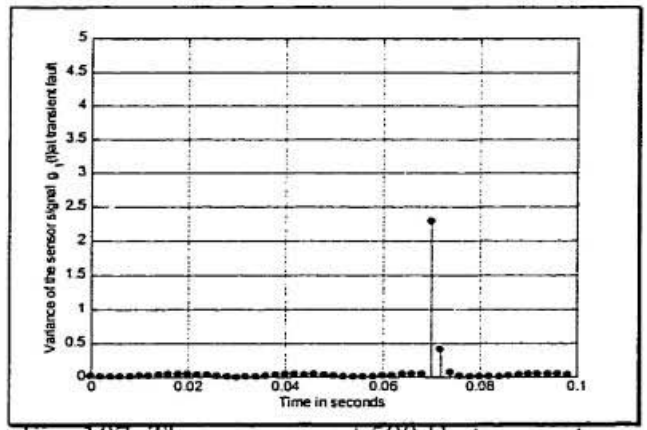


Fig. 107: The variances at 500 Hz transient on 1st test signal.

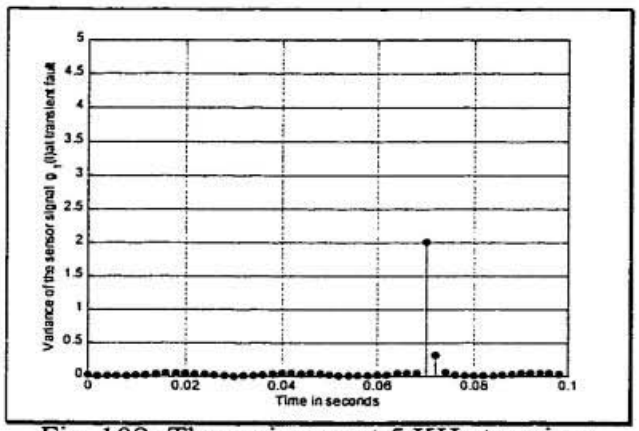


Fig. 108: The variances at 5 KHz transient.

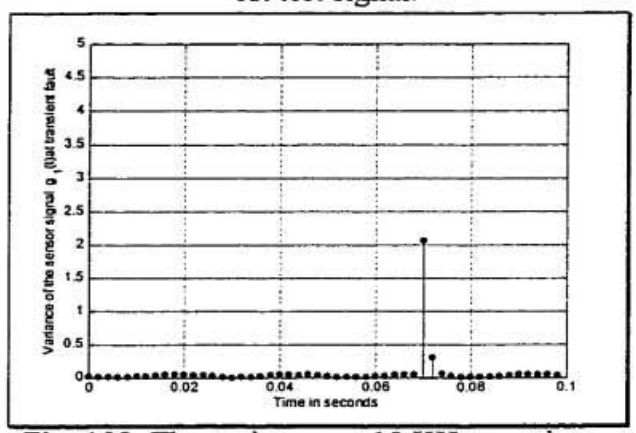


Fig. 109: The variances at 10 KHz transient.

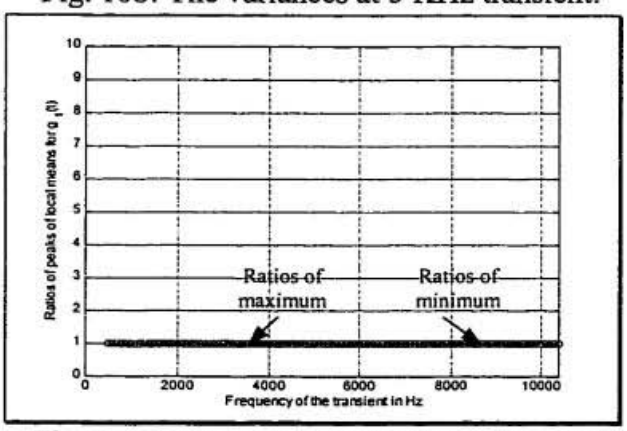


Fig. F.110:The ratios of the peaks of the local means at different transient frequencies.

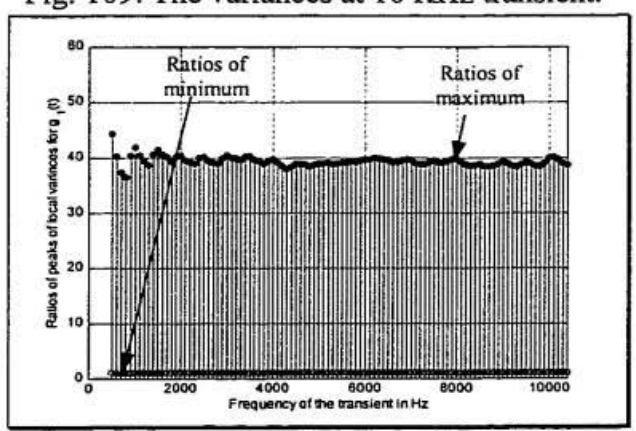


Fig. F.111:The ratios of the peaks of the local variances at different transient frequencies.

F.7.2 The Effect of Transient Faults of Different Frequencies on the Local Statistics of the Second Test Signal

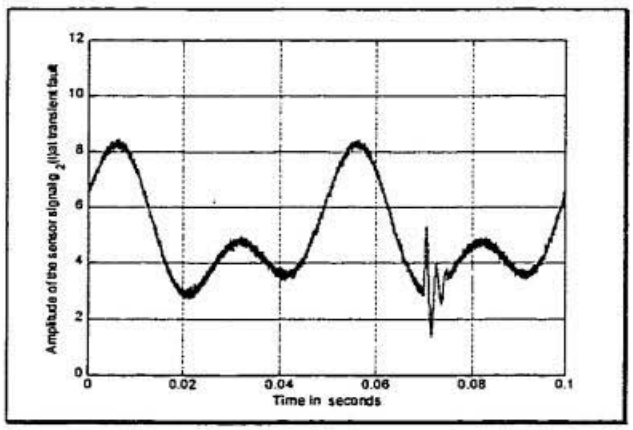


Fig. F.112: The 500 Hz transient on 1st signal.

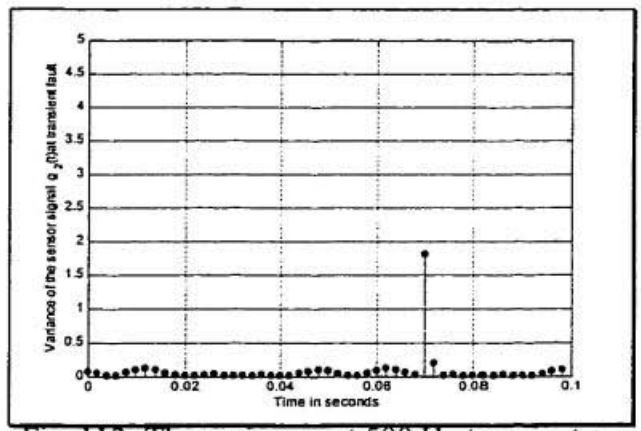


Fig. 113: The variances at 500 Hz transient on 1st test signal.

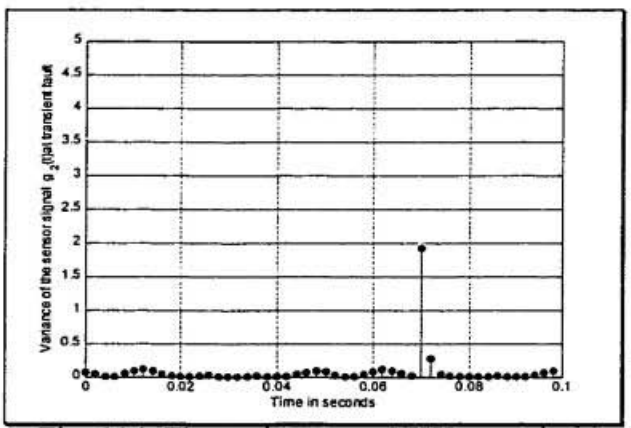


Fig. 114: The variances at 5 KHz transient.

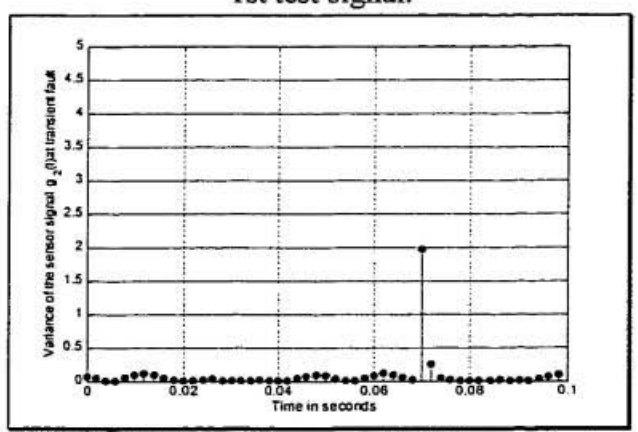


Fig. 115: The variances at 10 KHz transient.

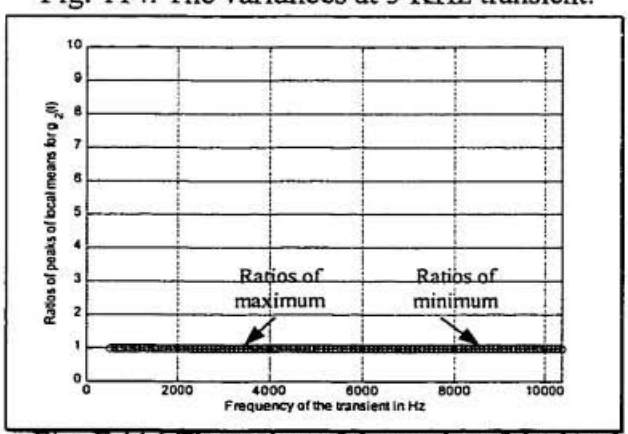


Fig. F.116:The ratios of the peaks of the local means at different transient frequencies.

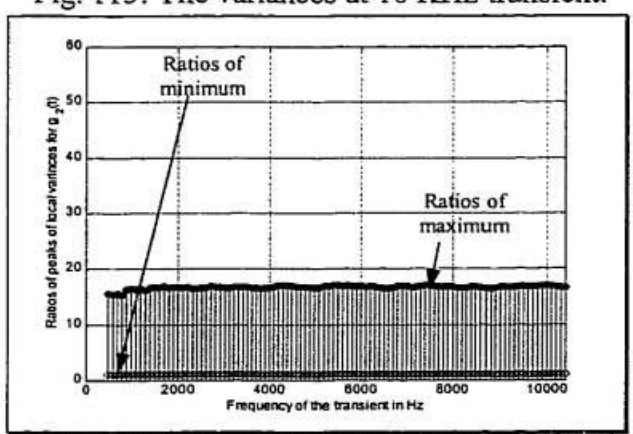


Fig. F.117:The ratios of the peaks of the local variances at different transient frequencies.

F.7.3 The Effect of Transient Faults of Different Frequencies on the Local Statistics of the Third Test Signal

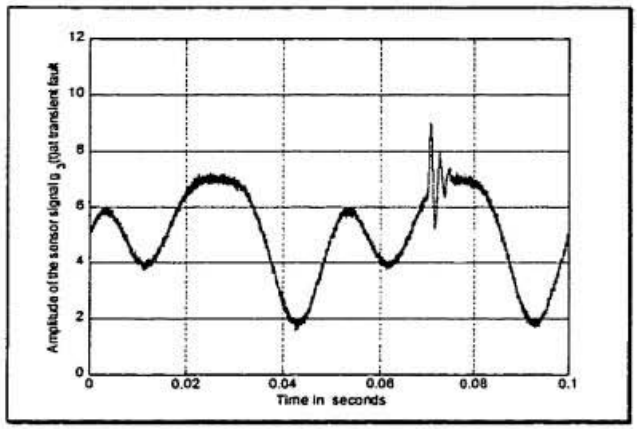


Fig. F.118: The 500 Hz transient on 3rd signal.

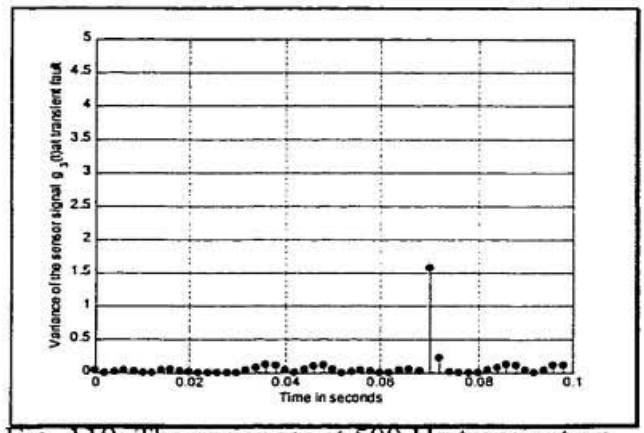


Fig. 119: The variances at 500 Hz transient on 3rd test signal.

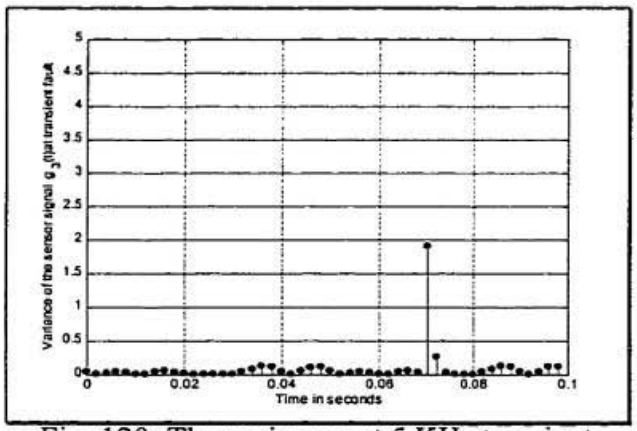


Fig. 120: The variances at 5 KHz transient.

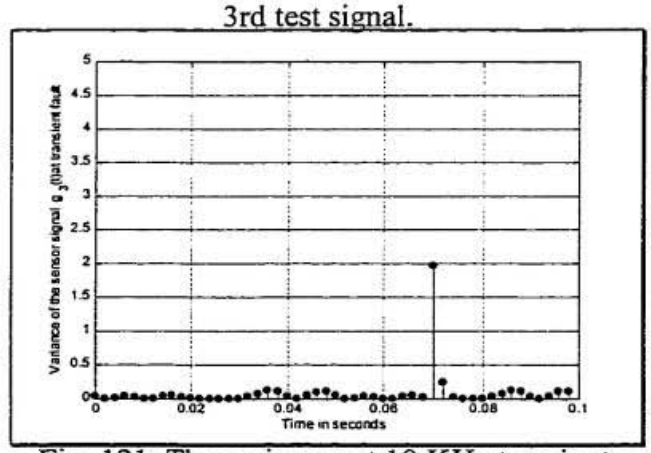


Fig. 121: The variances at 10 KHz transient.

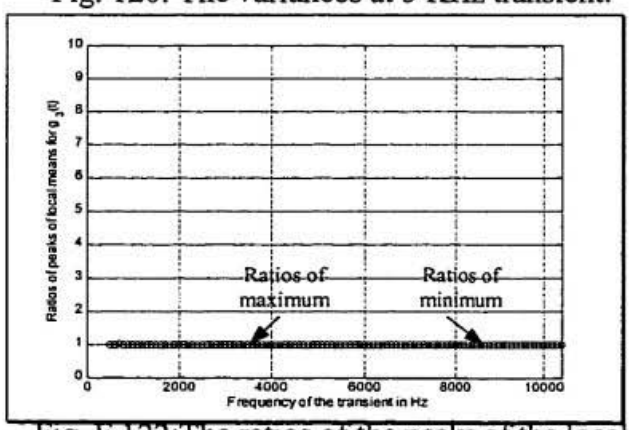


Fig. F.122:The ratios of the peaks of the local means at different transient frequencies.

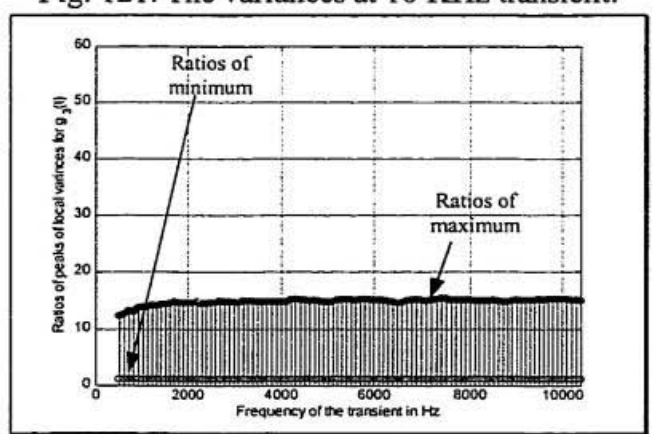


Fig. F.123:The ratios of the peaks of the local variances at different transient frequencies.

F.7.4 The Effect of Transient Faults of Different Frequencies on the Local Statistics of the Fourth Test Signal

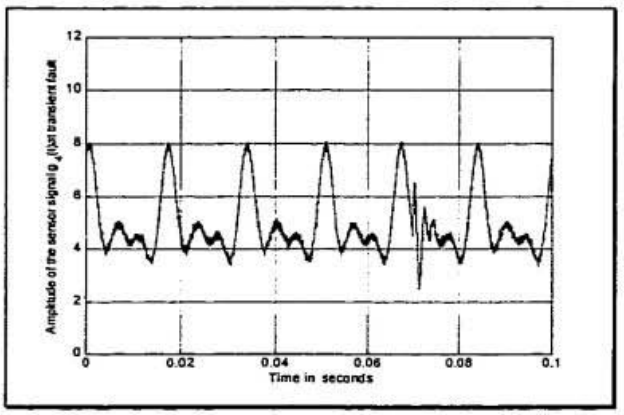


Fig. F.124: The 500 Hz transient on 4th signal.

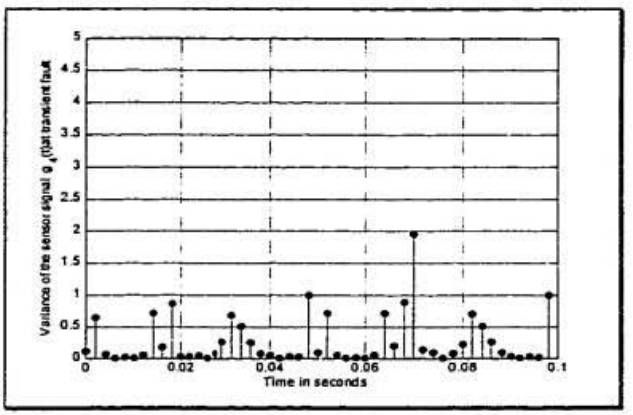


Fig. 125: The variances at 500 Hz transient.

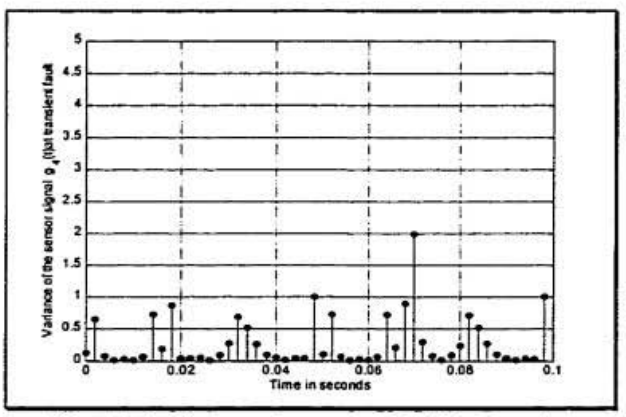


Fig. 126: The variances at 5 KHz transient.

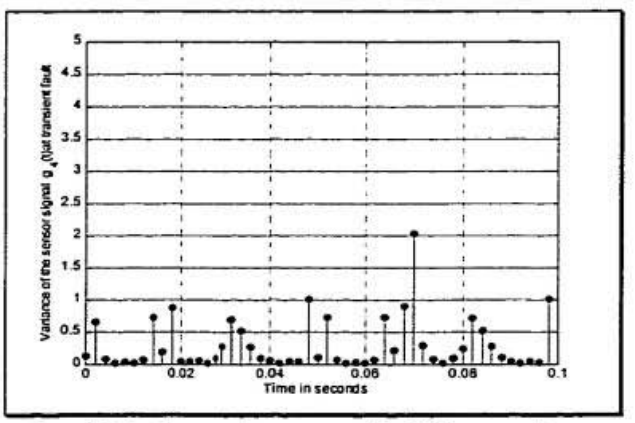


Fig. 127: The variances at 10 KHz transient.

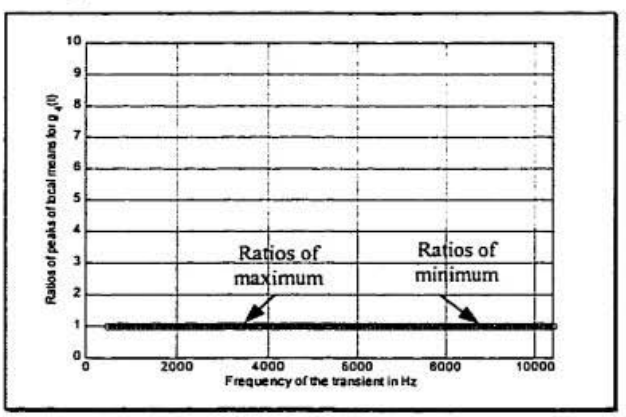


Fig. F.128:The ratios of the peaks of the local means at different transient frequencies.

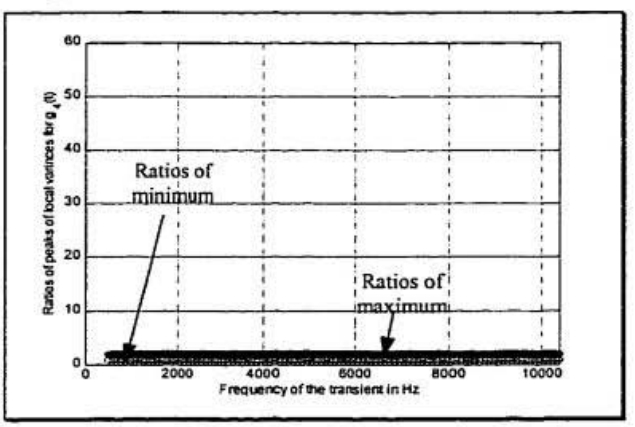


Fig. F.129:The ratios of the peaks of the local variances at different transient frequencies.

Appendix G

Performance of a Fault Tolerant Optical Sensor Using Triple Modular Redundancy

G.1 Introduction

An experiment was set up to study the potential of achieving sensor fault tolerance using triple modular redundancy. This study included the verification of the scheme to restore sensor data lost during fault clearance interval using parallel sensing technique proposed in Chapter 7.

Three photocells of same specifications were used to build the triple modular sensor system. The specifications of these photocells are shown in Table G.1.

Table G.1: The specifications of the optical sensors (photocells)

Sensitive area	Typical resistance at 10 Lux	Resistance at 100 Lux	Resistance at dark minimum	Max. applied voltage
0.45 cm^2	15 kΩ ±40%	3 KΩ	0.5 MΩ	200V (peak)

Each of these photocells was used to control the gain of an amplifier with the variation of its resistance due to the change of illumination level shown in Fig. G.1. The input voltage is a reference negative dc voltage which is amplified to the output as function of illumination level as shown in Eq.(G.1).

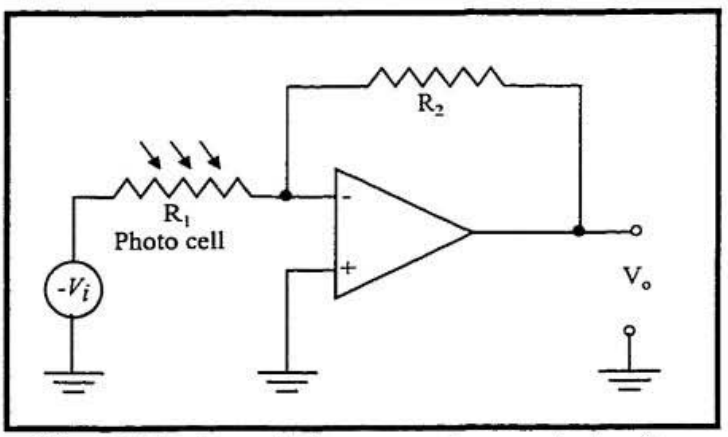


Figure G.1: An optical sensor whose output voltage level is function of illumination level.

$$V_0 = -\frac{R_2}{R_1}(-V_i) = \frac{R_2}{R_1}V_i \qquad \text{(G.1)}$$

Therefore, the output voltage of the optical sensor shown in Fig. G.1 is directly proportional to the illumination level (or inversely proportional to the value of resistance R_1).

G.2 Fault-Tolerant Optical Sensor Using Triple Modular Redundancy

A triple modular optical sensor system was realized connecting three optical sensors having same specifications to an analog multiplexer as shown in Fig. G.2. A microcontroller (MC68HC811E2FN) based single board computer was used for the detection of fault sensor and generation of control signals for the multiplexer to change the faulty sensor with fault-free one. A software running on the microcontroller based single board computer compares the signals from three sensors with each other, and, based on majority voting technique, detects the faulty sensor. If the fault channel is on as the multiplexer output, the channel is switched with a fault-free one by generating an appropriate control signal.

Artificial faults were generated by making a short circuit across the sensor, by disconnecting the sensor from the circuit, and by creating optical shadow on the sensor. Successful detection and followed by the switching of faulty channel with fault-free one as the output of the multiplexer was demonstrated. It should be noted that this triple modular redundancy based fault-tolerant optical sensor system couldn't detect faulty sensors if more than one sensor fail. It was noticed that the output signal dipped during fault clearance interval as shown in Fg.G.3. These dips were created to clear faults caused by artificial shadow on the photocells. The detection of these dips and the schemes for the minimization of the effect of these dips are explained in the following subsections.

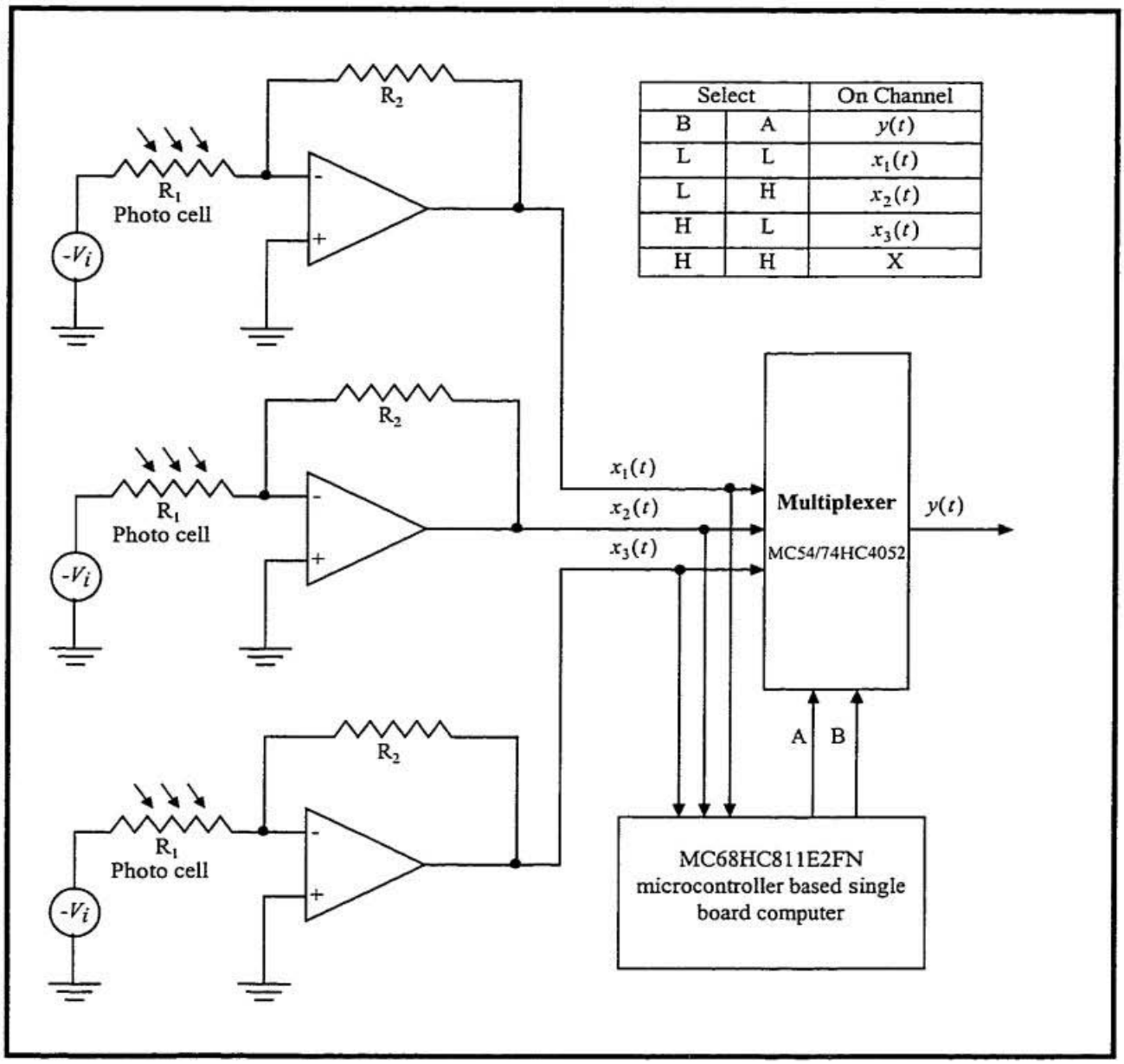


Figure G.2: An optical fault tolerant sensor using triple modular redundancy.

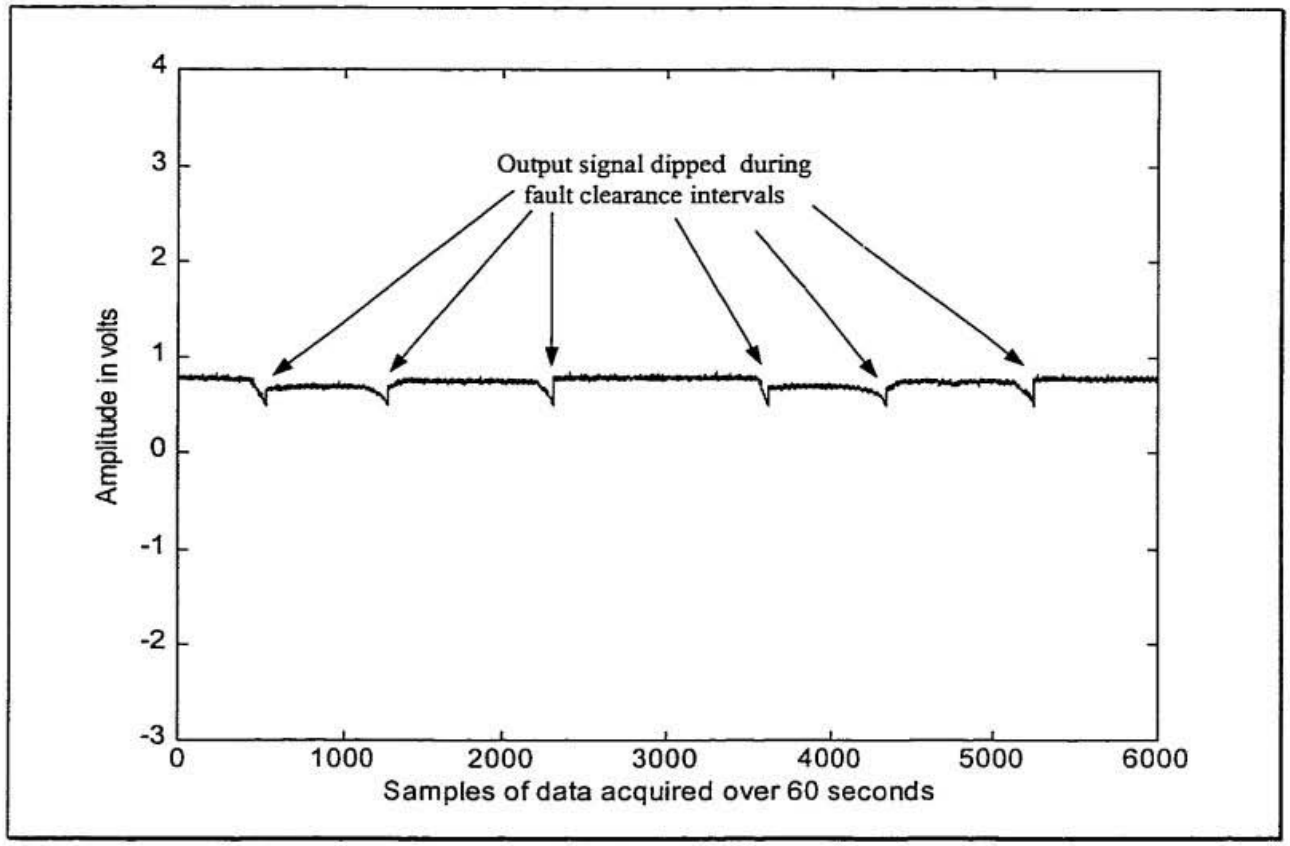


Figure G.3: The dips caused on the output signal from fault tolerant sensor module during fault clearance intervals.

G.3 The Detection of Fault Clearance Interval

The software running on the microcontroller board detects the faulty sensor and switches the output channel to the fault free sensor by generating control signal for the multiplexer. Therefore, the change in control signal can be used to detect the fault clearance interval as shown in Fig. G.4. The duration of fault clearance interval will depend upon particular situation. This window dimension can be calculated by comparing the signal value of the faulty sensor with that from a free one.

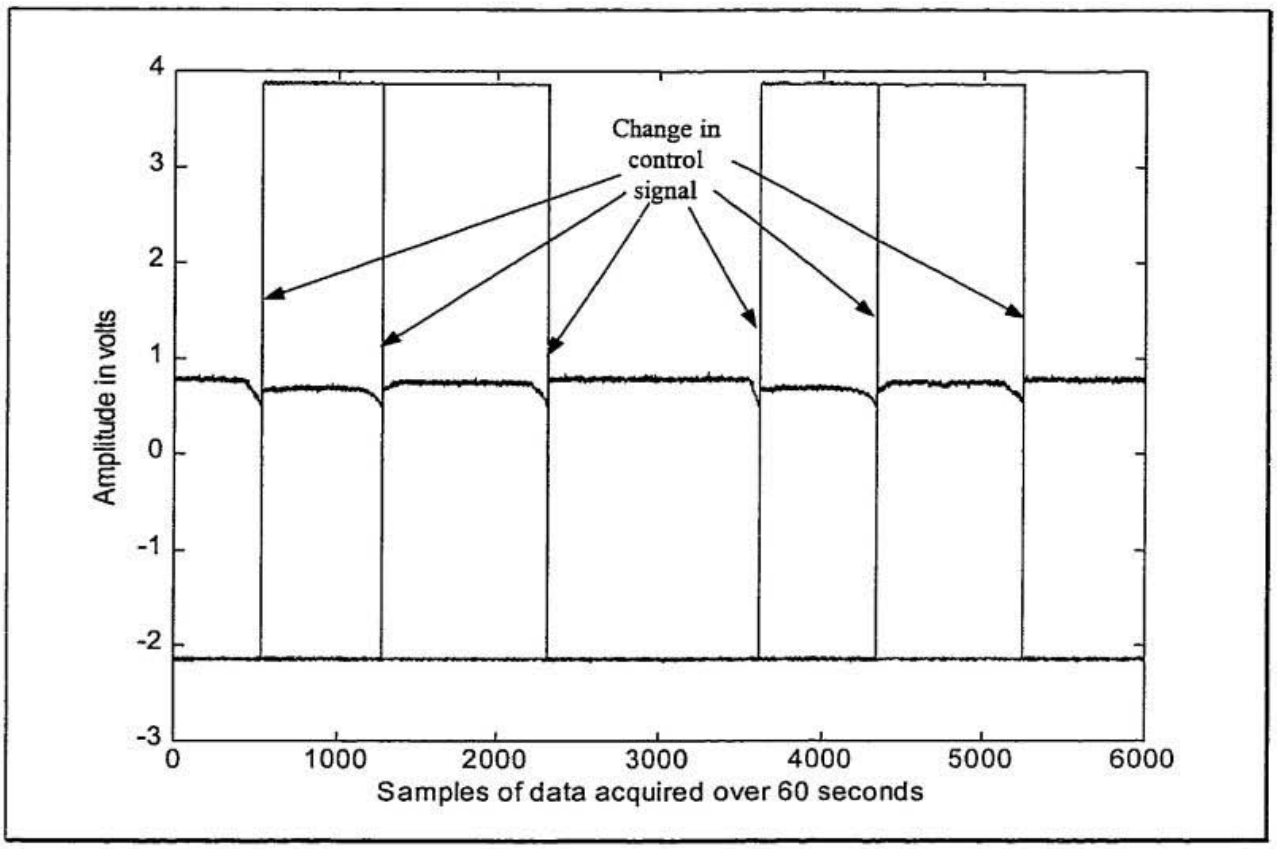


Figure G.4: The detection of fault clearance instances by monitoring the changes of the control signal sent to the multiplexer by the microcontroller board.

G.4 Minimization of the Effects of the Dips Caused during Fault Clearance Intervals

After the detection of fault clearance instances from the change of control signal, the data sample of faulty sensor around these instances can be compared with those of fault free sensor to measure the duration of the dips. Then for the duration of the fault clearance interval, data can be copied from the fault free sensor to replace the corresponding data samples of the output data stream to reduce the effect of these dips. This can be explained by removing the first dip. At the beginning, the multiplexer outputs the signal from the first sensor. The signals from the first sensor and the second sensor are shown in Fig. G.5 and G.6 respectively.

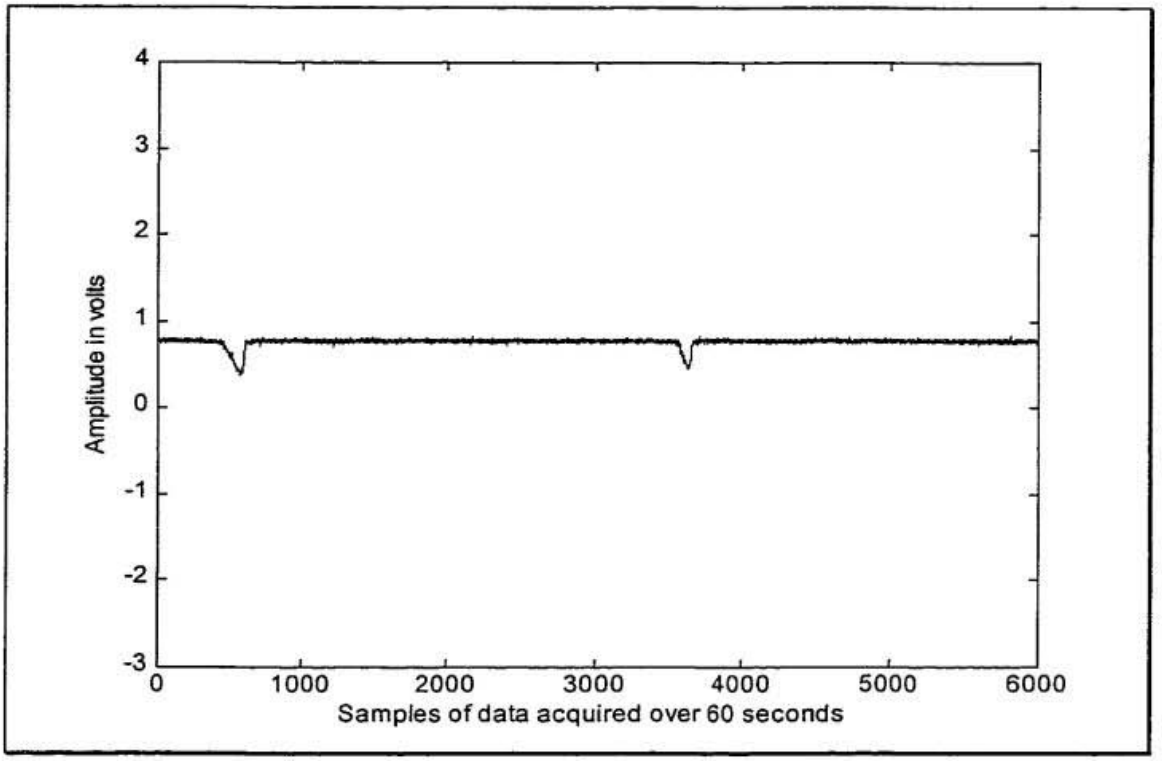


Figure G.5: The data stream from the first sensor.

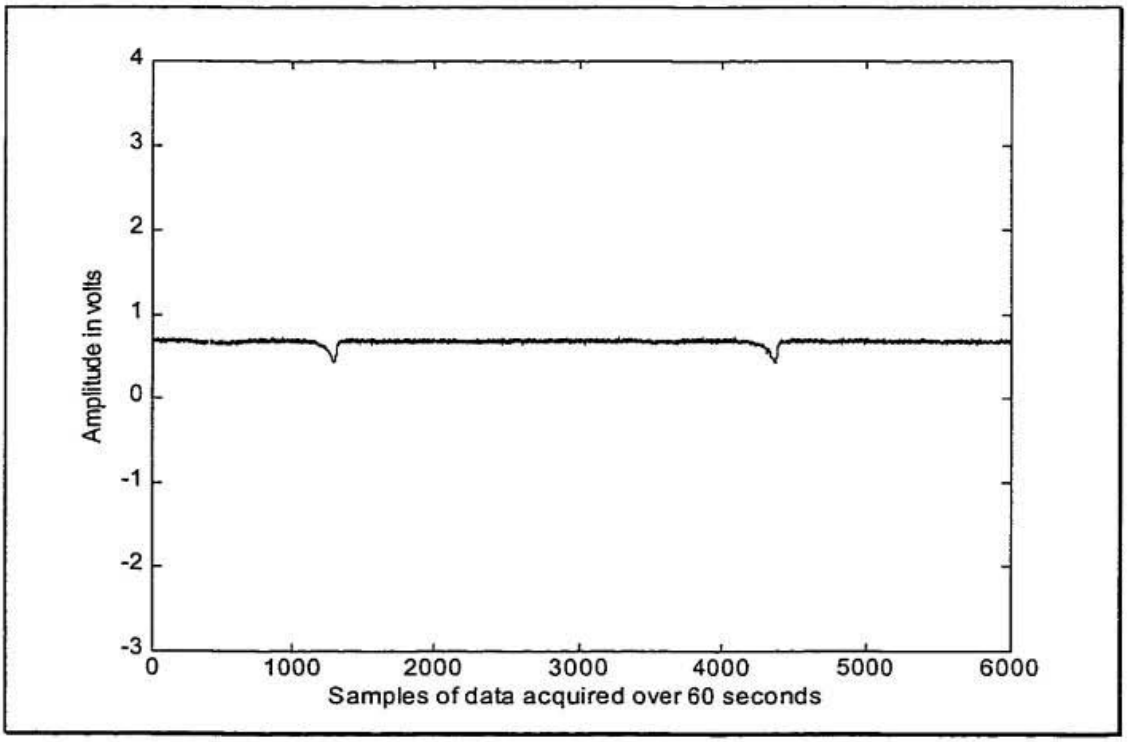


Figure G.6: The data stream from the second the sensor.

The location of the first dip caused on the output signal shown in Fig.G.4 corresponds to the first dip caused on the first signal as shown in Fig.G.5. At this location there is no dip caused on the second sensor signal. Therefore, if both the second sensor signal and the output signal of the multiplexer were recorded simultaneously, the first dip could have been removed by the scheme proposed in Chapter 7 as shown in Fig.G.7.

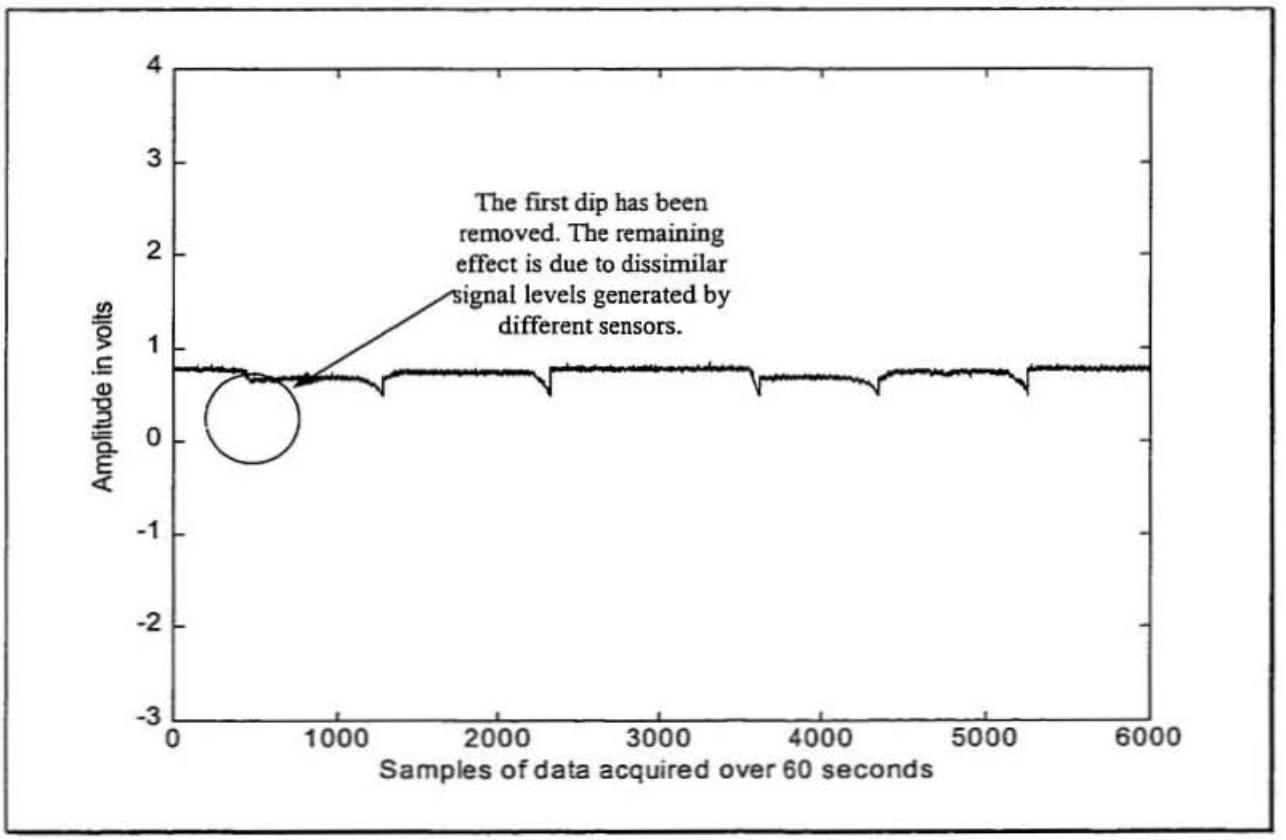


Figure G.7: The processed output signal from a fault tolerant sensor module after removal of the first dip.

In the similar way, the remaining dips can be removed and the processed output signal after the removal of all dips as shown in Fig. G.8.

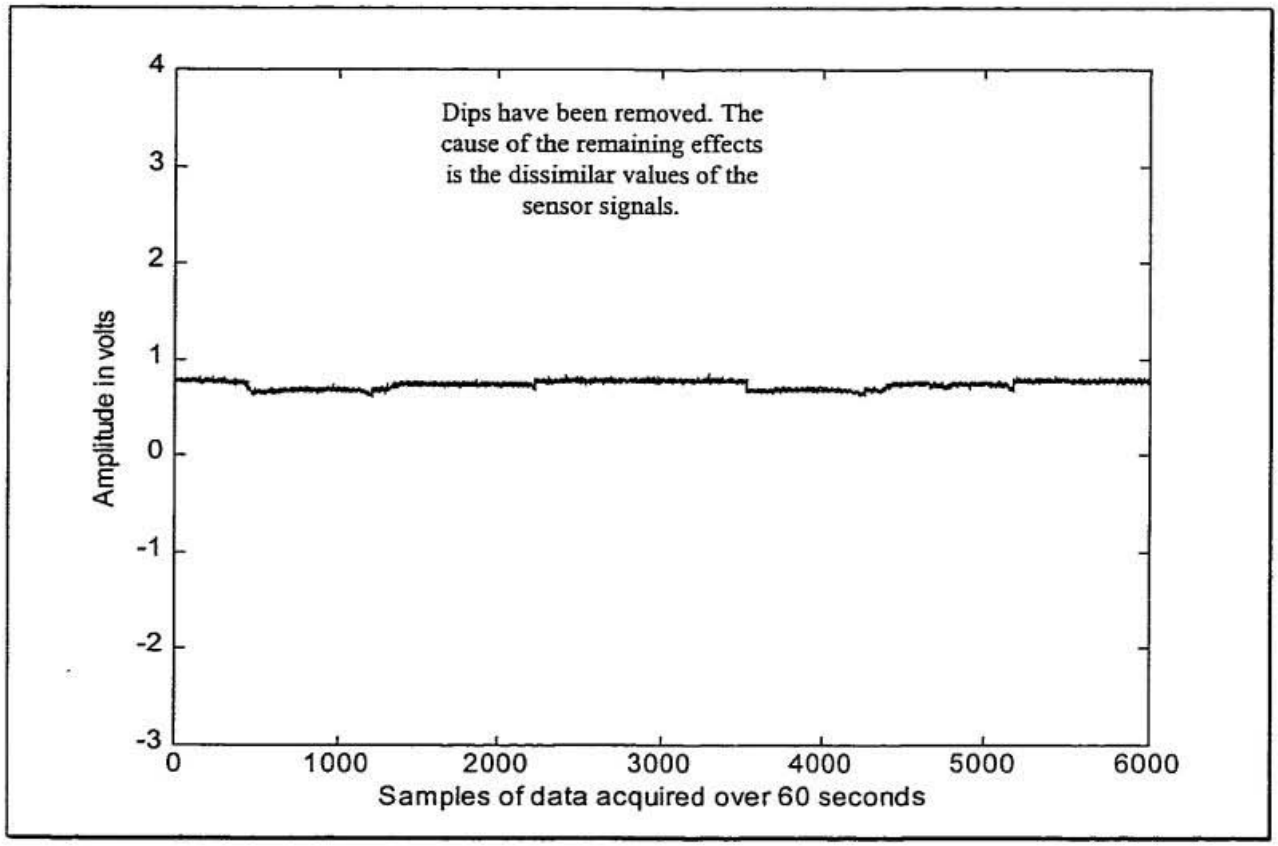


Figure G.8: The processed output signal of a fault tolerant sensor module with reduced effects for dips caused during fault clearance intervals.

The results of an experiment on fault-tolerant sensor system reported here explain that there is a potential to detect the faulty sensor and to replace it with fault-free one using triple modular redundancy. It has also been demonstrated that the effect of the dips caused on the output signal during fault clearance interval can be reduced by the parallel sensing scheme proposed in Chapter 7.