

PHYSICAL INTELLIGENT SENSORS

Pavan Bandhil, Sanjeevi Chitikeshi and Ajay Mahajan[♦]
Department of Mechanical Engineering and Energy Processes
Southern Illinois University Carbondale
Carbondale, IL 62901

Fernando Figueroa
NASA

John C Stennis Space Center
Technology Development and Transfer
Code HA30, Bldg. 8306
Stennis Space Center, Mississippi 39529-6000

Abstract

This paper proposes the development of intelligent sensors as part of an integrated systems approach, i.e. one treats the sensors as a complete system with its own sensing hardware (the traditional sensor), A/D converters, processing and storage capabilities, software drivers, self-assessment algorithms, communication protocols and evolutionary methodologies that allow them to get better with time. Under a project being undertaken at the NASA's Stennis Space Center, an integrated framework is being developed for the intelligent monitoring of smart elements. These smart elements can be sensors, actuators or other devices. The immediate application is the monitoring of the rocket test stands, but the technology should be generally applicable to the Integrated Systems Health Monitoring (ISHM) vision. This paper outlines progress made in the development of intelligent sensors by describing the work done till date on Physical Intelligent Sensors (PIS). The PIS discussed here consists of a thermocouple used to read temperature in an analog form which is then converted into digital values. A microprocessor collects the sensor readings and runs numerous embedded event detection routines on the collected data and if any event is detected, it is reported, stored and sent to a remote system through an Ethernet connection. Hence the output of the PIS is data coupled with confidence factor in the reliability of the data which leads to information on the health of the sensor at all times. All protocols are consistent with IEEE 1451.X standards. This work lays the foundation for the next generation of smart devices that have embedded intelligence for distributed decision making capabilities.

1. Introduction

Recent interest in smart sensor networks, and developments in technologies such as MEMS, microelectronics, nanotechnology, communication networks and distributed computing, have encouraged interest in the development of Integrated System Health Monitoring (ISHM) systems. Hence the need for intelligent sensors as a critical component for Integrated System Health Management (ISHM) is well recognized by now. An ISHM system is an example of an intelligent sensing system application. The purpose of such a system is to detect and measure certain quantities, and to use the information and knowledge obtained from the measured data,

[♦] Corresponding author

and any prior knowledge, to make intelligent, forward-looking decisions and initiate actions. Even the definition of what constitutes an intelligent sensor (or smart sensor) is well documented and stems from an intuitive desire to get the best quality measurement data that forms the basis of any complex health monitoring and/or management system. If the sensors, i.e. the elements closest to the measurand, are unreliable then the whole system works with a tremendous handicap. Hence, there has always been a desire to distribute intelligence down to the sensor level, and give it the ability to assess its own health thereby improving the confidence in the quality of the data at all times.

Sensors are a critical component of complex and sophisticated systems of today's technology and their role is ever evolving in the smart systems of tomorrow. General theories to treat intelligent sensor systems have been reported in the literature since the mid 80's [1-3]. Parallel work was done in industry where sensors have been developed with built in expert systems and look-up tables [4, 5]. These sensors, called smart sensors, were described as simple sensing devices with built-in intelligence. This intelligence included simple decision-making capabilities, data processing, conflict resolution, communications, or distribution of information. Figueroa and Mahajan [6] defined an autonomous sensor as a sensor that had an expert system with extensive qualitative tools that allowed it to evolve with time into a better and more efficient system. It differed, at least in philosophy, from the previous models by having a dynamic knowledge base as well as embedded qualitative and analytical functions that gave it a higher degree of operational independence, self-sufficiency and robustness. The underlying philosophy behind the autonomous sensor was probably closest to Henderson's [7,8] logical sensor models that also endeavored to give more problem-solving capabilities to the sensor, but still stayed away from any type of dynamic models.

DeCoste [9] described a system, called DATMI that dynamically maintained a concise representation of the space of local and global interpretations across time that were consistent with the observations. Each of the observations was obtained from a sensor, and therefore the number of observations was equal to the number of sensors in the control system. The truth of the observations and the validity of the sensors were obtained by cross-referencing with possible and impossible states of the system. DATMI was designed for a complete control system comprising of multiple sensors and actuators, and was the inspiration for the formalized theory called DATA-SIMLAMT (*Dynamic Across Time Autonomous - Sensing, Interpretation, Model Learning and Maintenance Theory*) which was designed for and is applicable to each sensor in the control system [10].

The key requirements of an advanced health monitoring system are that it should be able to detect damaging events, characterize the nature, extent and seriousness of the damage, and respond intelligently on whatever timescale is required, either to mitigate the effects of the damage or to effect its repair. These requirements have been discussed in some detail in earlier reports by Abbott [11-12]. According to Price, *et al* [13] a pure monitoring system is expected only to report damage rather than to formulate a response, but it is preferable that the ultimate objective of responding to damage be borne in mind from the outset. The statement of key requirements serves to sub-divide the problem as follows:

- i) Detection of damaging events: This requires some knowledge of the environment in which the system will be operating, the threats it will face, and the development of sensors and a strategy for using them to detect damage events well within the time

required for the system to respond. For events that require a rapid response, the best solution will often involve the use of passive, embedded sensors.

- ii) Characterization of the damage: This may or may not be a separate process from event detection. It may use different sensors, or the same sensors may be used in a different way. It is more likely to employ active sensors, which may be embedded in the structure or could be mobile and autonomous.
- iii) Prioritization of the seriousness of the damage: The first step of an intelligent response is to determine the seriousness of the damage in terms of its ability to compromise the mission of the vehicle, and consequently to determine the urgency (both relative and absolute) with which a response is required.
- iv) Identification of the cause of the damage: An intelligent system should be able to utilize data from a vast array of sensors to deduce information about the events that have occurred and the resulting damage, on a whole-of-vehicle basis.
- v) Formulation of the response: The nature of the response will depend on a number of factors such as the range of possible response mechanisms, the nature and severity of the damage, the available response time, etc. A response may consist of a sequence of actions. Major damage may demand an immediate "panic response", such as the rapid isolation of a whole section of the vehicle, followed by a more considered damage evaluation and repair strategy.
- vi) Execution of the response: In addition to repair, a holistic response may involve changes to the flight or operational characteristics of the vehicle, either to mitigate the effects of the damage or to assist in the avoidance of further damage.

The concept of PIS is nearly the same as described above [13] which states that it is most important to recognize that the purpose of a health monitoring system is to detect damage, so it must be able to function effectively in the presence of the damage. It should also be capable of distinguishing between (inevitable) failures of the system itself and damage to the vehicle structure. It must therefore be robust, adaptive, flexible, re-configurable and self-diagnosing. These requirements have led to the adoption of an approach based on a multiagent system (MAS) that consists of modular units that may be referred to as cells. These cells will not only form the physical structure of a vehicle, but will also have sensing, processing and communications capabilities.

The algorithms which run in the proposed physical intelligent sensors (PIS) are based upon the concept given by Maul [14]. The techniques published in [15-17] are also helpful in learning the implementation of the algorithms and the fundamental PIS structure. Some work has been done on the PIS to detect a single fault but there was no provision to detect multiple faults [18] hence this paper proposes to implement several event detection algorithms which can detect various kinds of faults on a sensor. In recent years it can be seen there is a large demand for integrated system health management systems (ISHM) [19-23] with applications in various fields like monitoring the structural health, nuclear power industry, ship harbor, furnace, turbine engine, thermal plants, etc.

This paper proposes the development of intelligent sensors as an integrated system approach. Over the years some work has been done in this area, but most of the work has been for customized applications. It is certainly now time to think of generic models for such types of sensors that can be quickly fitted in to any application. Under a project being undertaken at the Stennis Space Center (SSC), an integrated framework is being developed for the intelligent

monitoring of smart elements. These smart elements can be sensors, actuators or other devices. The immediate application is the monitoring of the rocket test stands, but the technology should be generally applicable to the ISHM vision. According to NASA as shown in Figure 1, from a perspective of intelligence or autonomy, the ISHM system should provide the following functions:

- System Monitoring
- Data Qualification
- Information Extraction
- Classification, Isolation and Diagnosis
- Mission Projection and Prognosis
- Communication and Information Transfer
- System Recovery and Response

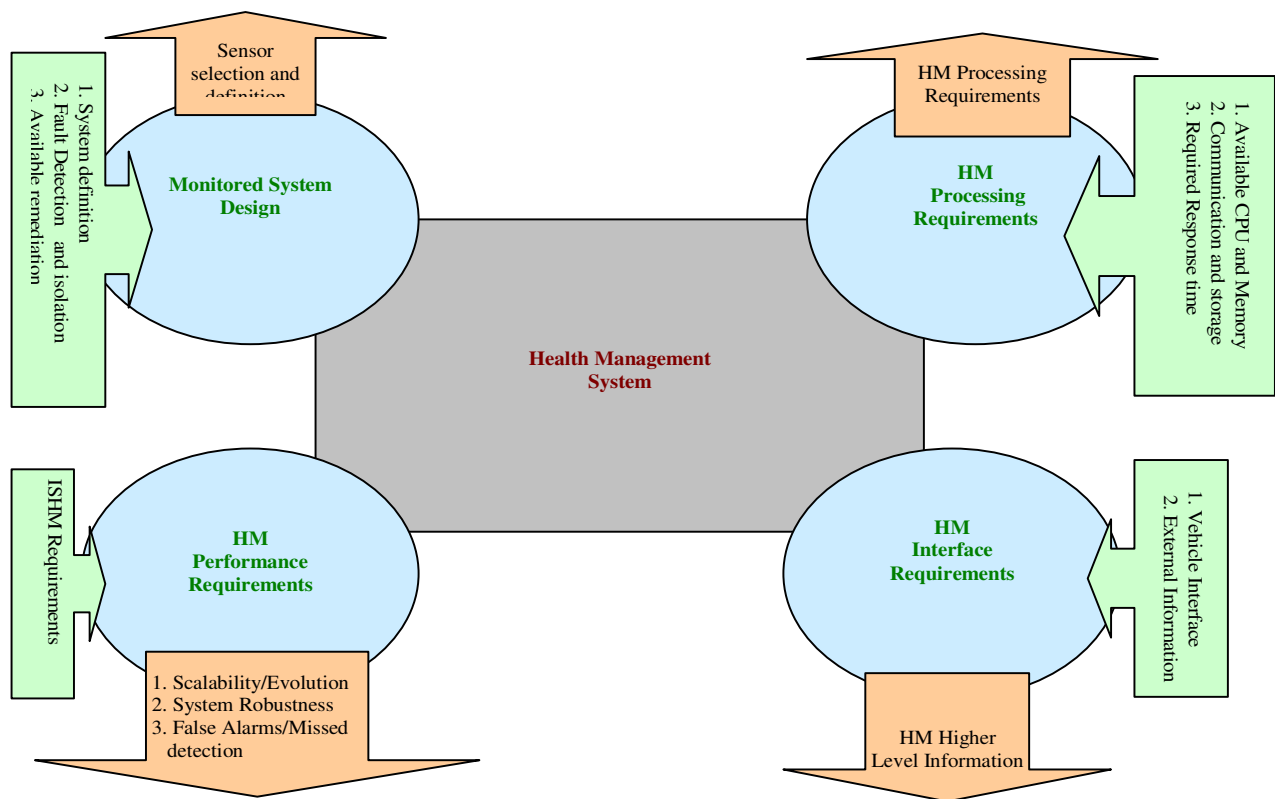


Figure 1: Overall structure of health management system

This paper also outlines progress made in the development of intelligent sensors by describing the following:

- A strategy for using a qualitative approach to process data and recognize problems in the data and/or in the health of the sensor itself.
- The introduction of a condition assessment sheet for each sensor that functions as a report card, and allows the system to make critical decisions during or after a run.
- The development of an integrated environment to run these intelligent sensors monitoring real processes.

2. Overall ISHM Structure

The governing Intelligent Systems Health Monitoring (ISHM) vision for an entire process needs to be in an environment conducive to embedded intelligence and decision making. Such an overall system for the a rocket test stand is being designed at Stennis Space Center, using the G2 environment (from Gensym, Inc.) but the scope of this work resides solely on the single sensor level. All of the external systems connect to G2 through a bridging program written in C\C++. The central system collects the data from the sensors and external programs and then applies it to the model of the system contained in its knowledgebase. Since this paper deals primarily in the single sensor realm a detailed discussion of the overall system will not be given.

Different types of sensors comprising a dynamic system will be instantiated as smart sensors that will fit within an object oriented integrated framework that uses embedded knowledge to monitor all the elements within the system. The concept of smart sensors may be extended to other types of elements as well as processes. Each element will have a specification sheet (SS) that will be fitted in to IEEE 1451.X standardized *Transducer Electronic Data Sheets* (TEDS), and a new entity called the *Condition Assessment Sheet* (CAS) that will be fitted into the *Health Electronic Data Sheets* (HEDS). The CAS shows the condition (or the health) of the element and its confidence in its own working for the duration of the operation. This "health" information is provided over and beyond the numeric output of the sensor. These smart elements will have decision making capability derived from embedded knowledge bases and their own intrinsic specification sheets.

Networks of elements with autonomous character will cooperate to perform as a system composed of a collection of processes, each managing a collection of sensors, actuators, and other components. The emphasis is on knowledge bases that support each element of the hierarchy and the relationships between them. A key feature of the proposed framework is the evaluation of condition for all elements performed both autonomously and using feedback from other higher-order elements. The proposed effort will develop and validate a hierarchical intelligent architecture composed of a system, processes, and sensors. Each element, as mentioned before, will be a *smart or intelligent* entity, where they possess the capacity to perform actions, assess those actions, and modify actions based on self assessment and external assessment of results by others. Implementing this level of intelligence involves embedding agents within each element that communicate, integrate, and adapt based on access to knowledge bases and autonomous learning algorithms.

The smart elements are being developed in the MATLAB environment which is very conducive for research purposes, while the system integration is being done using the G2 software, which is designed to handle complex intelligent systems. G2 software has been chosen since it offers the opportunity to develop layered behaviors analogous to the hierarchical autonomous architecture we seek to develop. SSC provides an ideal test bed for this development effort due to ready access to a broad range of rocket engine test stands, associated data acquisition systems, and archival data. In addition, there is a large experienced user base, which can be drawn from to develop, refine, and validate the architecture elements.

The ISHM is envisioned to have two types of sensors: Physical Intelligent Sensors (PIS) and Virtual Intelligent Sensors (VIS). The PIS are actual sensors with embedded microprocessors, while the VIS are virtual software-based sensors that function as PIS where it is physically impossible or uneconomical to replace existing sensors with PIS. This paper concentrates on the description of the PIS entities.

3. Physical Intelligent Sensor (PIS) Structure

The PIS or smart sensor is a combination of a sensing element, a data acquisition chip, a microprocessor and an Ethernet connection that allows one to directly connect the sensor to an Ethernet Bus. A PC is connected to the microprocessor to down load the software to the RAM and also debug the program. Once the software is running, this PC is removed and all the data is collected by a remote PC thru the Ethernet Bus. Several such smart sensors can be connected to the same Ethernet Bus and controlled by the single remote PC.

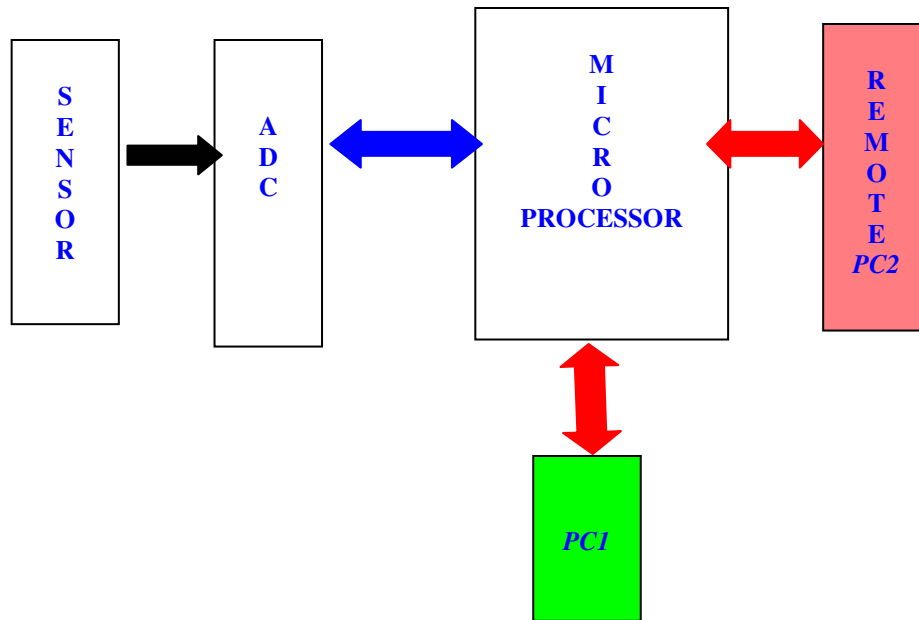


Figure 2: Block diagram of the PIS system

Figure 2 is the block diagram of the PIS system. As shown in the block diagram the first block is the sensor connected to the Analog to Digital Converter (ADC), where the analog data from the sensor is converted into digital data. The ADC is then connected to the microprocessor which is connected to a computer with a programming cable (RS-232). The program is downloaded onto the microprocessor. In this current work ADC7794 is being used for Analog to Digital conversion. It is a serial out converter. The microprocessor being used is the RCM3300 [24]. This microprocessor is compatible with Dynamic 'C' (8.61) software along with assembly language. Hence, one can do the software coding directly in Dynamic 'C' (8.61), thus reducing the complexity of the problem, and can see the results on the computer (a stdout window). One can also send the programmed data from the microprocessor to a remote system using a TCP/IP Protocols thru an Ethernet bus. The analog data from the sensor is given to the ADC. According to the specifications of the ADC, if the analog data from the sensor exceeds 5V then the analog voltage is scaled to 5V and is sent to the ADC. The same case is applied when the actual voltage is dropped below 0V i.e. dropped to a negative voltage, then the voltage is scaled to 0V and is sent to the ADC. The analog data is converted into digital data and is stored in the microprocessor's (RCM3300) Flash memory (512K) in the form of a file. This data forms the input for the event detection routines embedded in the PIS.

4. PIS Hardware Description

The hardware used in the PIS consists of an ADC7794, a microprocessor RCM3300 and a PC having windows 2000 and above operating system with a dynamic C 8.61 environment. Figure 3 shows the functional block diagram of the AD7794. The analog output from the sensor is given to the multiplexer and then this output is sent to the buffer. Then in order to amplify the signal from the sensor one has to pass this thru an operational amplifier due to the possibility of small strength signals. These pre-conditioned signals are sent to the ADC where these analog signals are sampled and are converted into digital signals which are then fed to the microprocessor. This data is processed using the smart software program. The digital data can be available at the (digital out) DOUT pin of the AD7794 chip. When the data is ready at the DOUT pin then the SCLK pin goes high indicating to the microprocessor that the data is ready. The microprocessor then accepts the data and makes the (digital in) DIN pin high indicating the data transfer is complete. This process continues until there is data available at the DOUT pin.

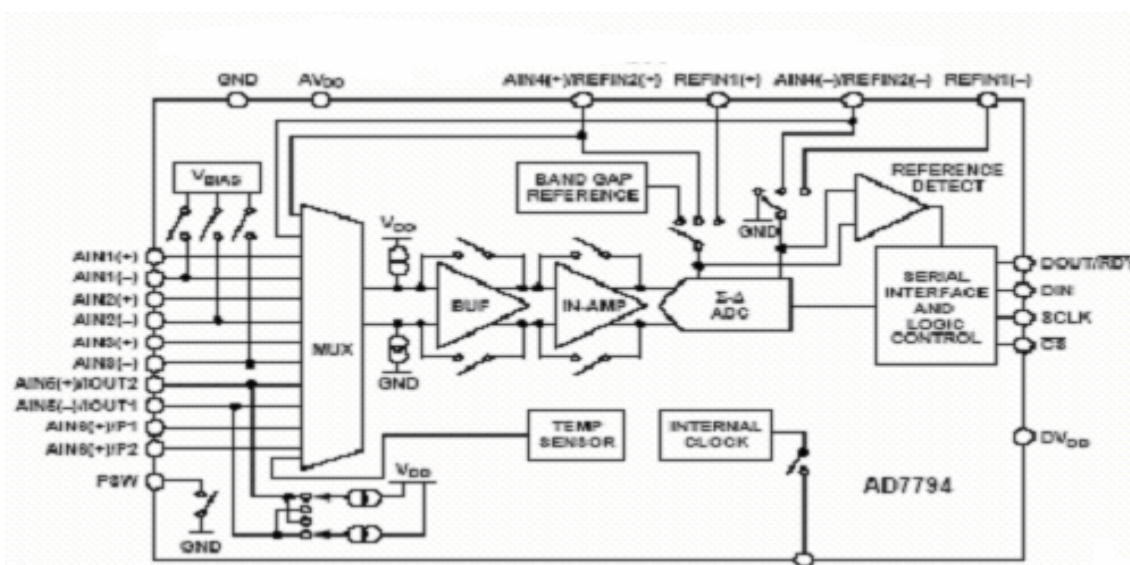


Figure 3: Functional block diagram of the AD7794

The SCLK has a Schmitt-triggered input, making the interface suitable for opto-isolated applications. The serial clock can be continuous with all the data transmitted in a continuous train of pulses. Alternatively, it can be a non continuous clock with the information being transmitted to or from the ADC in smaller batches of data. The internal clock can be made available at the Clock In/Clock Out pin. Alternatively, the internal clock can be disabled and the ADC can be driven by an external clock. This allows several ADCs to be driven from a common clock, allowing simultaneous conversions to be performed. The Chip Select Input is an active low logic input used to select the ADC. CS can be used to select the ADC in systems with more than one device on the serial bus or as a frame synchronization signal in communicating with the device. CS can be hardwired low, allowing the ADC to operate in a 3-wire mode with SCLK, DIN, and DOUT used to interface with the device. The main interface part in the smart sensor is the interfacing between the ADC and Microprocessor. The output data from the ADC is a 24bit serial data, so there are three pin connections between the ADC and the microprocessor. The following are descriptions of the pins:

- i) Interrupt pin from the ADC indicating that the data is ready at the ADC for transfer (SCLK pin from the ADC).
- ii) Data transfer pin (DOUT/READY pin from ADC).
- iii) Data acknowledgement pin from the microprocessor (DIN to the ADC).

When there is data ready at the ADC then the interrupt pin is high indicating that the data is ready for the microprocessor. When the microprocessor receives the interrupt signal from the ADC then the serial data is transferred to the microprocessor through the data transfer pin of ADC. When the data is transferred then the microprocessor sends a data acknowledgement signal to the ADC indicating the completion of the data transfer. As the data transfer includes serial/parallel data transfer, the clock of the microprocessor and ADC are synchronized, so that there is a minimal probability of data loss. The clock adjustment and the port pins (I/O) of microprocessor are controlled by the software (Dynamic 'C') in the microprocessor. The actual physical set-up of this interface connection is shown in Figure 4.

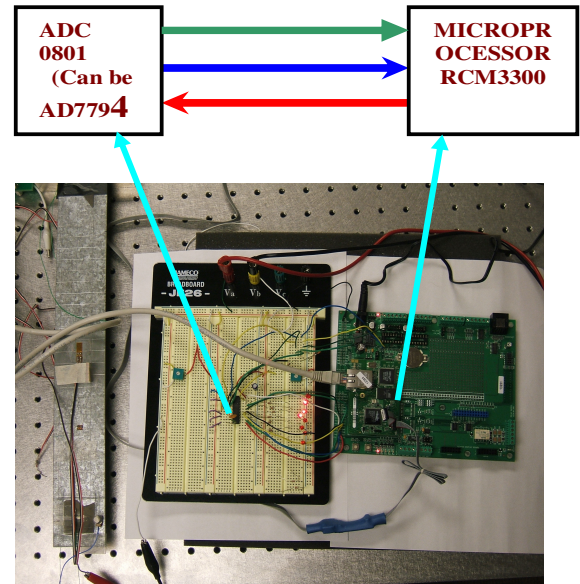


Figure 4: Interface between ADC and the Microprocessor

The microprocessor and the PC1 are connected by an RS232 programming cable. The code that has to be stored on the microprocessor has to be first written on PC1 and is then downloaded to the microprocessor. It is compiled and run by using the SRAM (or) FLASH (each of 512KB) of the microprocessor. There is a memory back up of 8MB in the form of serial flash on the RCM3300 core module to create files and to store data. This then has to be transferred to the remote PC2 for further analysis using TCP/IP protocols and an Ethernet cable connected to the microprocessor and the remote PC2. The physical set-up is shown in Figure 5.

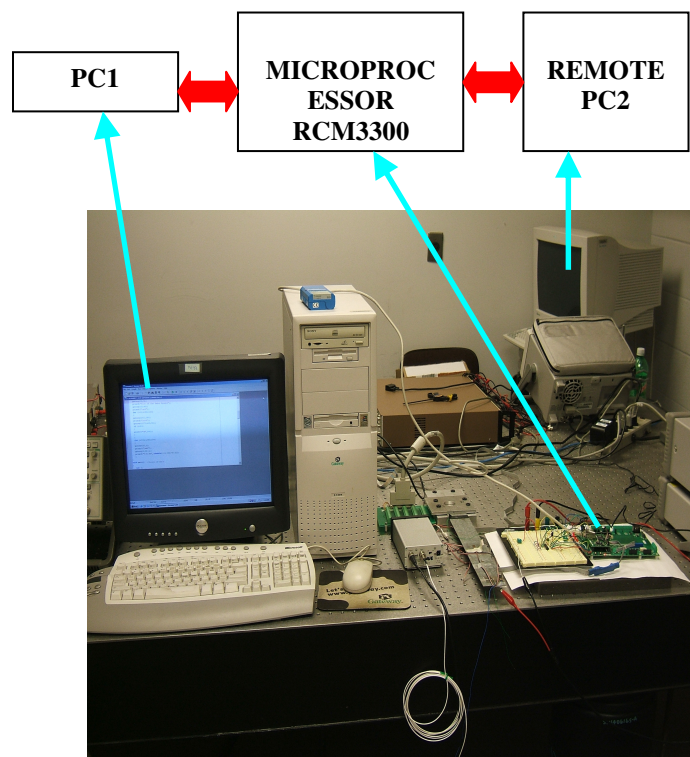


Figure 5: Sending results to the remote computer

Rabbit Memory Organization

Typical Rabbit systems have two types of physical memory: flash memory and static RAM memory.

a. Flash Memory: Flash memory in a Rabbit-based system may be small-sector type or large-sector type. Small-sector memory typically has sectors of 128 to 1024 bytes. Individual sectors may be separately erased and written. In large-sector memory the sectors are often 16K, 64K or more. Small-sector memory provides better support for program development and debugging while large-sector memory is less expensive and has faster access time. The best solution will usually be to lay out a design to accept several different types of flash memory including the flexible small-sector memories and the fast large-sector memories.

b. SRAM: Static RAM memory may or may not be battery-backed. If it is battery-backed it retains its data when power is off. Static RAM chips typically used for Rabbit systems are 32K, 64K, 128K, 256K or 512K. When the memory is battery-backed power is supplied at 2-3V from a backup battery. The shutdown circuitry must keep the chip select line high while preserving memory contents with battery power.

A basic Rabbit system has two static memory chips, one flash memory chip and one RAM memory chip. Additional static memory chips may be added. If an application requires storing a lot of data in flash memory, another flash memory chip can be added (e.g. an external XD-Flash Card) creating a system with three memory chips, i.e. two flash memory chips and one RAM chip.

There are four main memory segments:

1. The Root Memory Segment: The root memory segment has a typical size of 24K. The larger the root memory segment smaller the data segment and vice-versa. Root memory segment address zero is always mapped to a 20-bit address zero. Usually the root memory segment is mapped to flash memory since root code and root constants do not change except when the system is reprogrammed. It may be mapped to RAM for debugging, or to take advantage of the faster access time offered by RAM. The root memory segment holds a mixture of codes and constants. C functions or assembly language programs that are compiled to the root memory segment are interspersed with data constants. Data constants are inserted between blocks of code. Data constants defined inside a C function are put before the end of the code belonging to the function. Data constants defined outside of C functions are stored as encountered in the source code. Except in small programs, the bulk of the code is executed using the extended memory segment. But code in the root memory segment operates slightly faster. Also the root memory segment has special properties that make it better for some types of code.
2. The Data Segment: The data segment is mapped to RAM and contains C variables. Typically it starts at 8K or above and ends at 52K (0xCFFF). Data allocation starts at or near the top and proceeds in a downward direction. It is also possible to place executable code in the data segment if it is copied from flash to the data segment. This can be desirable for code that is self modifying, code to implement debugging aids or code that controls write to the flash memory and cannot execute from flash. In some cases RAM may require fewer wait states so code executes faster if copied to the RAM.
3. The Stack Segment: The stack segment normally is from 52K to 56K (0xD000-0xDFFF). It is always mapped to RAM and holds the system stack. Multiple stacks may be

implemented by defining several stacks in the 4k space or by remapping the 4K space to different locations in physical RAM memory, or by using both approaches. For example, if 16 stacks of 1k length are needed then 4 stacks can be placed in each 4k mapping and 4 different mappings for the window can be used.

4. The Extended Memory Segment: This 8K segment is from 56K to 64K (0xE000-0xFFFF) and is used to execute extended code. It is also used by routines that manipulate data located in extended memory. While executing code the mapping is shifted by 4K each time the code passes the 60K point. Up to a megabyte of code can be efficiently executed by moving the mapping of the 8K window using special instructions (long call, long jump and long return) that are designed for this purpose.

The following is the process followed for the building of code.

- The programmer edits the code in the Dynamic 'C' IDE, and saves the source file in a text format.
- The Dynamic 'C' IDE compiles the code. If needed, the programmer can compile from command line parameters. Unlike most other development environments, Dynamic 'C' prefers to compile every source file and every library file for each build. There is an option that allows the user to define precompiled functions.
- There is no separate linker. Each build results in a single binary file (with the ".BIN" extension) and a map file (with the ".MAP" extension).
- The Dynamic 'C' IDE downloads the executable binary file into the target system using the programming cable.

Figure 6 shows the program flow for the PIS and the execution of the event detection algorithms.

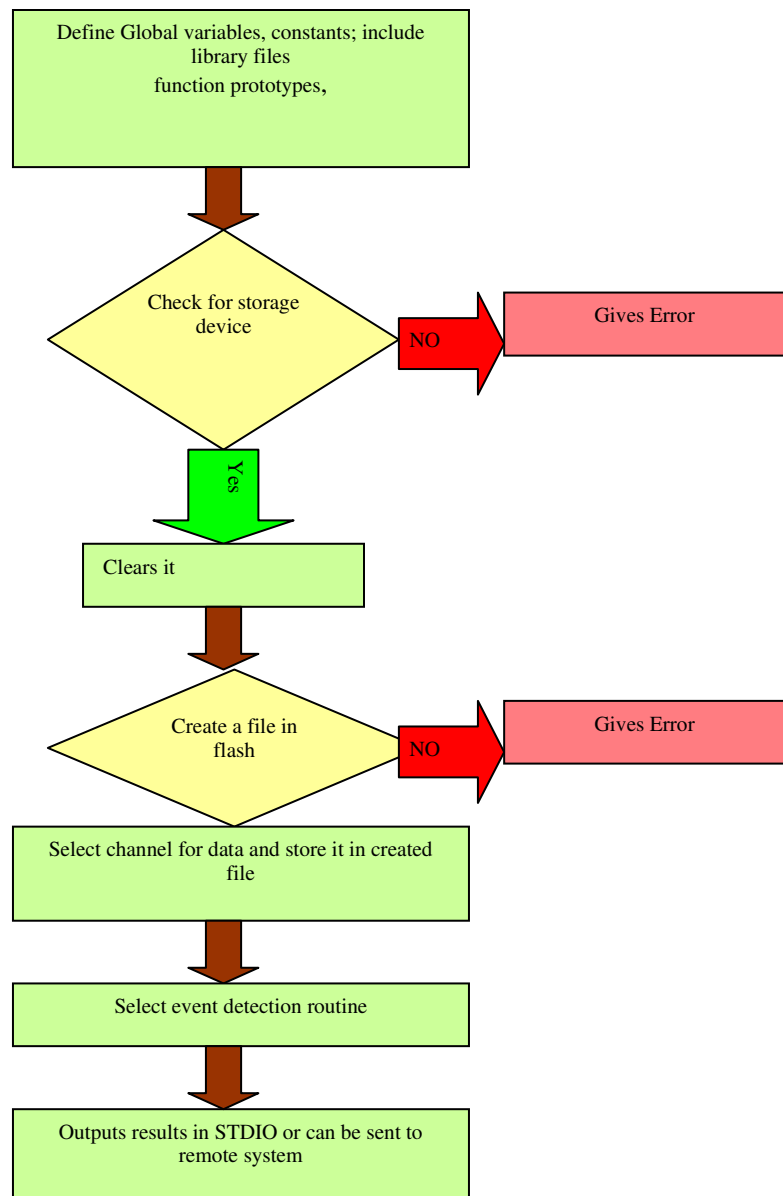


Figure 6: Program flow for the PIS

5. PIS Implementation and Execution

The PIS as described in the previous sections has been implemented with a thermocouple and has the capability to work in real time. The following routines obtained from NASA's Glenn Space Center have been implemented on the PIS:

- A. Level Shift
- B. Flat
- C. Noise
- D. Spike
- E. Drift

To demonstrate the PIS capabilities an artificial data set was created with all the above events/anomalies as shown in Figure 7. Letters A-E signify the routines recognizing the events/anomalies and the corresponding PIS outputs are shown in this section.

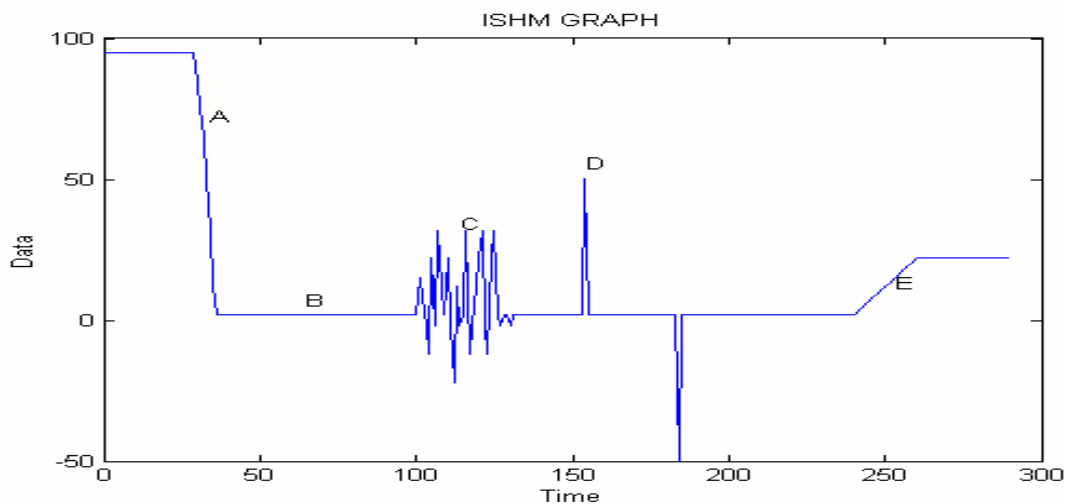
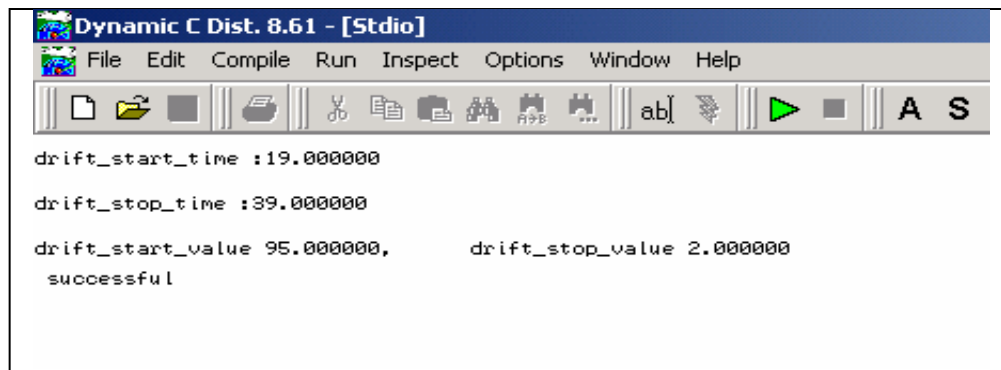


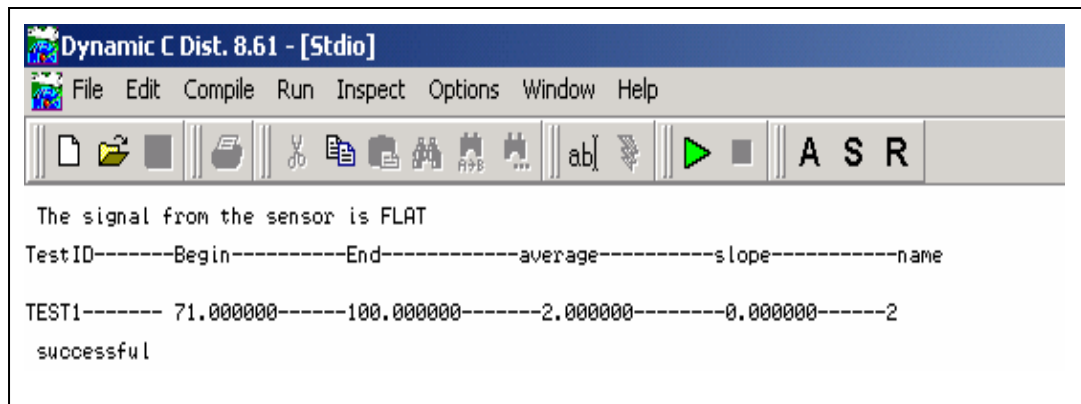
Figure 7: Test data with multiple events detected by the PIS

A: Level Shift



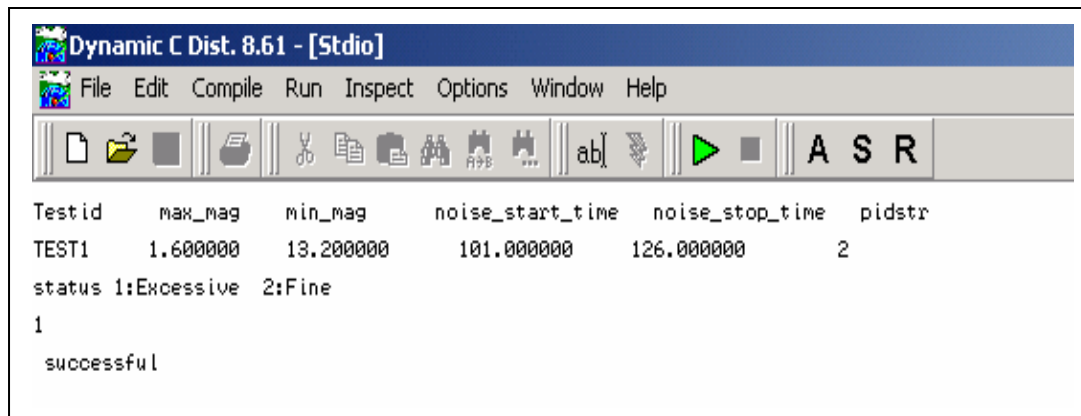
The PIS output shows that the level shift has occurred at start time 19.0 (time units) with a previous stable value 95.0 (temperature units) and stopped at 39.0 (time units) reaching a stable value of 2.0 (temperature units).

B: FLAT



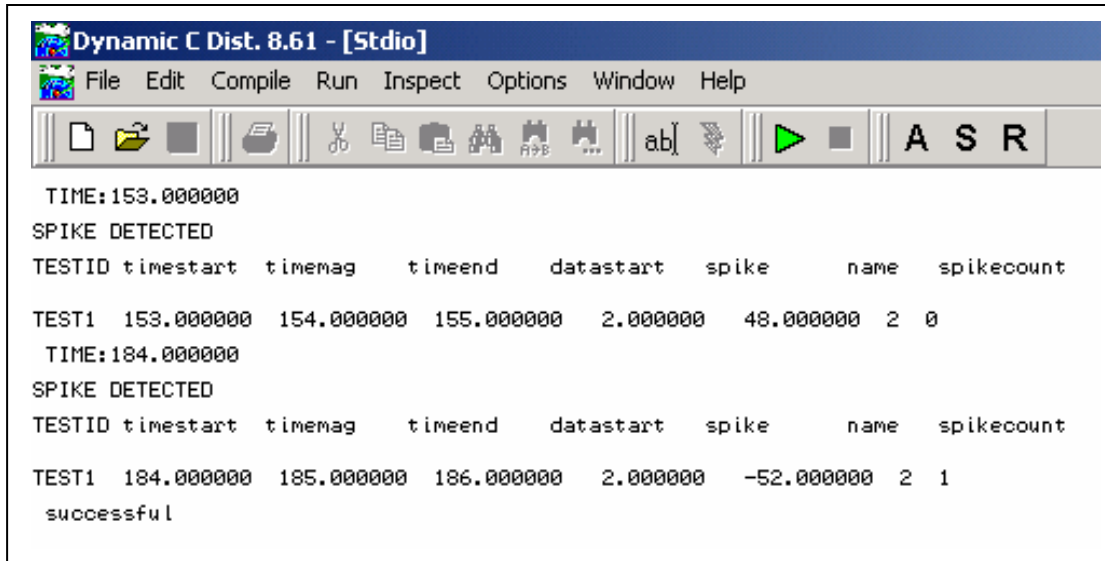
The PIS output shows that during the interval 71.0 (time units) to 100.0 (time units) the sensor output is flat. This means that the sensor is dead or is not powered up. It can be seen from the graph that there is no change in the sensor data in the observed period, in other words there is even an absence of noise, hence the suspicious nature of the sensor output.

C: NOISE



The PIS output shows that excessive noise was detected at 101.0 (time units) and ended at 126.0 (time units). During this period the maximum average noise was of magnitude 13.2 (data units) and minimum average was 1.6 (data units). The “noise” event detection routine returns a status of 1 or 2, where 1 is “excessive noise” and 2 is “fine.”

D: Spike

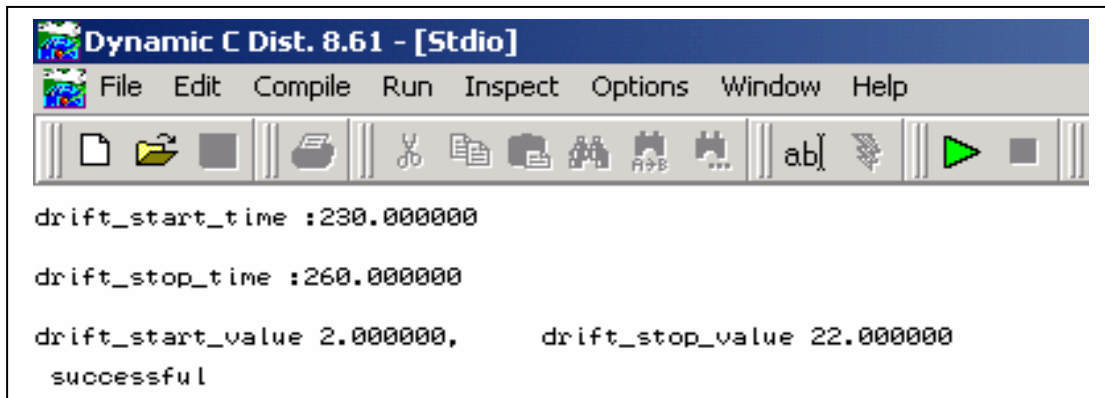


```
Dynamic C Dist. 8.61 - [Stdio]
File Edit Compile Run Inspect Options Window Help
[Icons] [ab] [Play] [A] [S] [R]

TIME:153.000000
SPIKE DETECTED
TESTID timestart  timemag   timeend   datastart  spike      name  spikecount
TEST1  153.000000  154.000000  155.000000  2.000000  48.000000  2    0
TIME:184.000000
SPIKE DETECTED
TESTID timestart  timemag   timeend   datastart  spike      name  spikecount
TEST1  184.000000  185.000000  186.000000  2.000000  -52.000000 2    1
successful
```

The PIS output shows that there were two spikes detected in the given stream of data. The first spike was detected at time 153.0 (time units) which lasted till 155.0 (time units). The time stamp for the start of the spike is 2.0 (data units) and the spike had a magnitude of 48.0 (data units) i.e. $50.0 - 2.0 = 48.0$. Similarly a second spike was detected at 184.0 (time units) with a magnitude of -52.0 (data units).

E: Drift



```
Dynamic C Dist. 8.61 - [Stdio]
File Edit Compile Run Inspect Options Window Help
[Icons] [ab] [Play] [A] [S] [R]

drift_start_time :230.000000
drift_stop_time  :260.000000
drift_start_value 2.000000,      drift_stop_value 22.000000
successful
```

The PIS output shows that a drift was detected that started at 230.0 (time units) from a value of 2.0 (data units) and ended at 260.0 (time units). It settled at a final new value of 22.0 (data units).

Conclusions

This paper has presented the development of intelligent sensors as part of an integrated systems approach, i.e. one treats the sensors as a complete system with its own sensing hardware (the traditional sensor), A/D converters, processing and storage capabilities, software drivers, self-assessment algorithms, communication protocols and evolutionary methodologies that allow them to get better with time. Under a project being undertaken at the NASA's Stennis Space Center, an integrated framework is being developed for the intelligent monitoring of smart elements. This paper outlines progress made in the development of intelligent sensors by describing the work done till date on Physical Intelligent Sensors (PIS). The PIS discussed here consists of a thermocouple used to read temperature in an analog form which is then converted into digital values. A microprocessor collects the sensor readings and runs numerous embedded event detection routines on the collected data and if any event is detected, it is reported, stored and sent to a remote system through an Ethernet connection. Hence the output of the PIS is data coupled with confidence factor in the reliability of the data which leads to information on the health of the sensor at all times. All protocols are consistent with IEEE 1451.X standards. This work lays the foundation for the next generation of smart devices that have embedded intelligence for distributed decision making capabilities.

Acknowledgements

The authors would like to acknowledge the support of NASA for funding this work under Grant NNS04AB796. The authors would also like to acknowledge the contributions of researchers at Rowan University for providing the smart sensor prototype and NASA Glenn Research Center for providing the event detection routines.

References

1. Ghani, N., 1988, "Sensor integration in ESPRIT," *IFAC Proceedings*, Karlsruhe, FDR, pp. 323-328.
2. Pinkava, J., 1989, "Towards a theory of sensory robotics," *Robotica*, Vol. 8, pp. 245-256.
3. Lozano-Perez, T., Mason, M. T., and Taylor, R., 1984, "Automatic synthesis of fine motion strategies for robots," *International Journal of Robotics Research*, **3**, No. 1, , pp. 2-24.
4. AbdelRahman, M. and Smith M. L., 1991 "The Impact of AI On Sensing Technology," *SENSORS*, pp. 16-22.
5. Studt, T., 1994, "Smart Sensors Widen Views on Measuring Data," *R&D Magazine*, pp. 18-20.
6. Figueroa, F. and Mahajan, A., 1994, "Generic Model of an Autonomous Sensor," *Mechatronics*, Vol. 4, No. 3, pp. 295-315.
7. Henderson, T and Shilcrat, E., 1984, "Logical Sensor Systems," *Journal of Robotic Systems*, **1**(2), pp. 169-193.
8. Henderson, T., Hansen, C. and Bhanu, B., 1985, "The Specification of Distributed Sensing and Control," *Journal of Robotic Systems*, **2**(4), pp. 387-396.
9. DeCoste, D., 1991, "Dynamic Across-Time Measurement and Interpretation," *Artificial Intelligence*, Vol. 51, pp. 273-341.
10. Mahajan, A. and Figueroa, F., 1995, "Dynamic Across Time Autonomous - Sensing, Interpretation, Model learning and Maintenance theory (DATA-SIMLAMT)," *Mechatronics*, Vol. 5, No. 6, pp. 665-693.

11. Abbott, D., 2001. "Development and Evaluation of Sensor Concepts for Ageless Aerospace Vehicles", Report 1, CSIRO TIP Report no. TIPP 1516.
12. Abbott, D., "Development and Evaluation of Sensor Concepts for Ageless Aerospace Vehicles", Report 2, CSIRO TIP Report no. TIPP 1517.
13. Price, D.C., Scott, D.A., Edwards, G.C., Batten, A., Farmer, A.J., Hedley, M., Johnson, M.E., Lewis, C.J., Poulton, G.T., Prokopenko, M., Valencia, P. and Wang, P., 2003. "An Integrated Health Monitoring system for an Ageless Aerospace Vehicle" *Proceedings of the Structural Health Monitoring 2003: From Diagnostics & Prognostics to Structural Health Management*, ed. Fu-Kuo Chang, DEStech Publications (Lancaster, PA), pp. 310-8.
14. Maul, W., 2005 "ISHM/ NASA Session - Work and Technology at NASA-Algorithms for Intelligent Elements" IEEE Sensors for Industry Conference.
15. Wang, P., Valencia, P., Prokopenko, M., Price, D.C. and Poulton, G.T., 2003. "Self-Reconfigurable Sensor Networks in Ageless Aerospace Vehicles," *Proceedings of the 11th International Conference on Advanced Robotics (ICAR-03)*, Coimbra, Portugal.
16. Lovatt, H., G. T. Poulton, D. C. Price, M. Prokopenko, P. Valencia, P. Wang. 2003. "Self-Organising Impact Boundaries in Ageless Aerospace Vehicles," In *Proceedings of the 2nd International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS '03)*, Melbourne, Australia, July 2003.
17. Newman, R.M. and Gaura, E.I., 2003 "Using very large arrays of intelligent sensors" *Proceedings of the 2003 IEEE/ASME international Conference on Advanced Intelligent Mechatronics* (AIM 2003).
18. Gaura, E.I., and Newman, R.M., 2003, "Intelligent sensing: Neural Network based health diagnosis for sensor arrays," *Proceedings of the 2003 IEEE/ASME international Conference on Advanced Intelligent Mechatronics* (AIM 2003).
19. Minh-Duc L.E., Thi-Hoang-Hoa, Thi-Huong V.U., Duc-Vang L.E., Thi-Huong, 2003 "An Automatic Control System Applying Decoupling Control Methodology for Ship Harbor Maneuvers," *Proceedings of the 2003 IEEE/ASME international Conference on Advanced Intelligent Mechatronics* (AIM 2003).
20. Seliger, G., Kross, U. and Buchholz, A., 2003, "Efficient Maintenance Approach by On-board Monitoring of Innovative Freight Wagon Bogie," *Proceedings of the 2003 IEEE/ASME international Conference Of Advanced Intelligent Mechatronics* (AIM 2003)
21. Changfan, Z., Jing, H., Yonghong, L. and Yeqing, C., 2004, "Intelligent Temperature Control of Ignition Furnace in Sintering Machine," *Proceedings of 2004 IEEE Conference on Cybernetics and Intelligent Systems*, Singapore.
22. Frank L. Greitzer, Lars J. Kangas, Kristine M. Terrones, Melody A. Maynard, Bary W. Wilson, Ronald Pawlowski, Daniel R. Sisk, and Newton B. Brown, 1999, "Gas Turbine Engine Health Monitoring and Prognostics", *Proceedings of the International Society of Logistics (SOLE) 1999 Symposium*, Las Vegas, Nevada.
23. McDonald, J.R., McArthur, S.D.J. and Burt, G.M., 2001, "Intelligent system applications for power system control and management," *Computing & Control Engineering Journal*, Volume 12, Issue 2, April 2001, pp.85-91.
24. Hyder, K. and Perrin, B., 2005, "Embedded Systems Design Using the Rabbit 3000 Microprocessor," Butterworth-Heinemann College, ISBN: 0750678720.