Robotica (1999) volume 17, pp. 635–647. Printed in the United Kingdom © 1999 Cambridge University Press

# ViSIAr – A virtual sensor integration architecture

Nigel Hardy* and Aftab Ahmad Marooof†

(Received in Final Form: June 5, 1999)

**SUMMARY**

Virtual sensors (software abstractions to support programming of sensor use) have been shown to have software-engineering benefits. A sensor integration system is required to support them. We examine the general requirements of such systems and consider the important design requirements. An idealised architecture, ViSIAr, is proposed to serve as a framework for designing and constructing them. Illustrative examples are provided.

KEYWORDS: ViSIAr; Virtual sensor; Sensor integration

## 1. INTRODUCTION

We report an idealised architecture, which forms the basis for the design and development of Sensor Integration (SI) systems to implement the *virtual sensor* concept. Virtual sensors provide support for sensory robot programming where complex systems must be built, modified and maintained frequently. They are software entities, which abstract sensing functionality and support the programming of sensor use by hiding hardware dependencies and by providing consistent and reconfigurable software modules.

Virtual sensors offer the following general advantages:

1. De-coupling of lower level interfacing and higher level use of sensors leads to easier maintenance and upgrading. Hardware changes need not imply extensive software changes.
2. Consistent and relevant application programmer interfaces (APIs) for sensor use lead to simpler and less error prone code.
3. API routines can enforce correct use of sensors through compile time or run time checking.
4. Safe and efficient concurrent use of sensors can be achieved without special top level design. The feasibility of concurrent use of two dynamically created virtual sensors based on the same transducer can be checked when they are created.
5. The consistent APIs and the reliability of concurrent use allow creation of derived sensors using hierarchies of more primitive ones.

The requirements of systems to support virtual sensors are analysed and developed further in this paper.

* Department of Computer Science, University of Wales Aberystwyth, Penglais, Aberystwyth, Ceredigion, SY23 3DB (UK) e-mail: nwh@aber.ac.uk
† FAST Institute of Computer Science, 4 Bazar Road, G-6/4, Islamabad (Pakistan)

## 2. BACKGROUND

The idea of abstracting physical sensor details to enable sensing device independence has attracted interest in the robotics community. A number of systems have been suggested, each with its own emphasis on a particular aspect of the problem.

Inspired by the work on logical graphics input devices, Henderson proposed sensor data abstraction in the form of logical sensors.[1–4] The emphasis was on the resilience to internal failure of the sensor system, provided through knowledge based switching between redundant information sources for the same *logical sensor output*. Weller[5] proposed a finer granularity of abstraction for sensor modules with emphasis on looking for the cause of sensor failure and trying to recover if possible, before the radical step of discarding a particular sensor module. Erdmann[6] describes a system for automatically designing software sensors from the specification of the robot's task. Duffy *et al.*[7] use *virtual sensors* to abstract physical sensing details and to provide a uniform interface for data fusion and to produce best estimates for the world model. Hewlett-Packard's Visual Engineering Environment (VEE) is a graphical programming language, which offers an environment particularly suited to test and measurement. Visual programming objects provide simple and consistent interfaces to devices. A number of interface technologies, including HP-IB and standard serial lines are supported. Use of legacy interface code is also facilitated. Through specialised drivers, a range of devices (both local and on the Web) are made available with standard interfaces for incorporation into VEE programs.

The proposals described here have developed from early attempts at standardised sensor control[8] and modular development of sensor software.[9] The virtual sensor concept as described here was developed to provide sensing functions at a level more suitable for robot programming.[10] This led to sensor control abstraction as well as sensor value abstraction.[11] The virtual sensing concept emphasises the analysis of the use of sensory information in robot operation and subsequent design of virtual sensing functions to support this with the most convenient interface. In the InFACT project[12] the virtual sensor concept was expanded and applied in the context of an integrated flexible assembly machine.[13–15] The real-time aspects of virtual sensors have been investigated[16] and are not considered here. The concept of virtual sensors and proposals for the design and development of sensor integration systems based on them have been explored more recently by Aftab Ahmad[17] and elements of this work are described here.

In a number of systems concentrating on higher level facilities, regular treatment of low level sensing is con-

https://doi.org/10.1017/S0263574799001915 Published online by Cambridge University Press

sidered. In developing SI systems, it has been the intention to support such use. Multi-sensor fusion[18,19] tackles the problem of combined readings from a number of sensors to give more accurate, complete and robust information. A range of techniques is available and under investigation. Mobile robotics has seen some of the more sophisticated sensing regimes,[20] though often not involving large numbers of sensors. Research has led to a range of architectures for control. Many have a sensing paradigm as a central feature. Perceptual Schemas[20,21] closely match the sensing requirements of action with sensing modules which answer these requirements in an appropriate way. At LASS-CNRS[22] an architecture supports high level perception based on a logical robot system which reduces the hardware dependency of the overall system. The Polly system[23] provided simple abstractions such as *open-left?* and demonstrated both concurrent use of sensing devices and sharing of processing steps and perceptual measurements. "Adaptive switching of specialised systems" was also suggested. In the Xavier system[24,25] low level motion reports such as "translation and rotation derived from wheel encoders" are distinguished from sensor reports such as "data from the sonar sensors that indicate whether the robot is in a corridor junction". Higher level functions are insulated from details of sensors. A sensor interpretation component turns the continual motion of the robot into discrete motion reports and uses an occupancy grid method to turn the raw sonar data into sensor reports of high level features such as walls and doorways. Monitoring for exceptions, differences between nominal and observed situations, is supported. The Saphira architecture[26] puts sensors to a variety of uses in terms of the paradigms of the consumer modules. Reactive behaviours take (possibly pre-processed) data from sensors. Strong internal representations are also used to de-couple control from the problems of interpreting noisy data.

The context in which we set SI systems is one of a set of sensors and actuators co-ordinated by a central computer. The architecture is not specified and might be a single computer with many devices interfaced to it; a distributed system using LAN communications technology; or some intermediate configuration. The central computer, or more specifically a particular program running on it, is known as the *supervisor*. The nature of this program is again not tightly specified. It could be equivalent to the many "traditional" robot programming language systems in the style of *VAL*[27] or *AML*.[28] Alternatively, knowledge based approaches including planners in the style suggested by systems such as Handley[29] and RALPH[30] could be employed. The supervisor could be a programming system in which applications are implemented or it could be a purpose built program for a particular application. It has been our intention to support the widest possible variety of supervisor paradigms.

This work has been carried out in the context of manipulator robots, and more specifically for application in Flexible Assembly Systems (FAS). This is an area where an application exists for this type of technology and it offers a specific and challenging focus for the work. It is expected, however, that the methods will find applications more widely. Fields such as continuous process control, home automation and the development of aids for the disabled may also benefit from it. Application is sought in mobile robotics.[31]

A detailed model of a virtual sensor is given elsewhere.[17,31] Here it is sufficient to consider it as an abstract device supporting a particular sensing functionality. The functionality is realised through sensor value abstraction and sensor control abstraction. Virtual sensors can be dynamically defined and used to provide sensory information in an abstract form to assist actuation control. A virtual sensor possesses a specific overall character or behaviour with a defined life cycle that determines legal sequences of commands it can receive.

To ensure good de-coupling while enforcing correct use of sensors and allowing hierarchical construction of sensors and systematic handling of failures, it is important to design a well defined set of commands for the supervisor to use. ViSIAr does not prescribe the form of these: it could be a communications protocol, a procedural API or other language specific mechanism. For simplicity we refer to this as the supervisor *language*, without any implied loss of generality.

The rest of this paper is structured as follows. Section 3 discusses the sensing requirements of an SI system while Section 4 considers the design requirements placed on it by its role and use. Section 5 describes the idealised architectural model for design and implementation that incorporates these considerations. Section 6 provides two illustrative examples of the benefits of the use of the techniques.

## 3. SENSOR INTEGRATION SYSTEM SENSING REQUIREMENTS

The functional requirements of a SI system must be derived from the needs of the supervisor. We consider first the general role of an SI system in providing appropriate facilities and then the specific sensor usage it must support.

### *3.1 The sensor abstraction gap*

The actuators, the associated feeding mechanisms and the parts themselves all introduce change and uncertainty in a FAS. The sensors, based on a variety of transduction mechanisms and with a range of accuracies, repeatabilities, data rates, reliabilities and relevances, are used to measure aspects of the system. In most cases, the data produced by the sensor is not readily usable for control. It must be processed to some degree, ranging from simple scaling or thresholding through noise reduction to sophisticated statistical techniques and sensor fusion. In addition, the process of obtaining the data will vary due to differences of computer interfacing as well as intrinsic differences in the sensors. The module within the FAS which consumes the data must typically become involved not just in the decision to obtain the data and in its use but also in the practical details of obtaining it. There is therefore a *gap of abstraction* between the sensor and the consumer of the data. There is a gap between the physical sensing and sensing usage. This gap can lead to poor engineering. If it is bridged by including extensive and specialised code in each

https://doi.org/10.1017/S0263574799001915 Published online by Cambridge University Press

consumer module, the complexity of the modules is increased (with associated problems of cost and reliability) and it represents poor de-coupling since changes to either the sensing mechanism or to the design of the module's control algorithm will often not be possible in isolation. Specialised code in each module is often the solution and in relatively long lived, static or small systems it can be cost effective. It is however a barrier to flexibility and to the reliable development of larger systems.

The gap is not static. Improvements in sensor technology and the development of *smart sensors*, which include significant local flexibility and data processing, mean that "sensors" as seen by the system integrator offer data at higher levels of abstraction than previously. On the other hand, more sophisticated FAS, larger systems and the incorporation of planning and other high-level modules place increasingly sophisticated demands on the sensor system. A gap remains therefore.

SI systems are designed and built to bridge the gap. Rowland and Nicholls[9] describe how a sensing system forms a virtual machine that lies between sensing devices and the supervisor and provides sensor knowledge in the form required. Any useful attempt to bridge the gap must consider and accommodate diverse and changing requirements on both sides.

The rationale of a SI system is therefore to deliver information at the level and in the context of where it is to be consumed and the details of physical sensing and processing of data are hidden from the supervisor. This approach can alleviate problems both of scale (i.e. large numbers of sensors) and of diversity (i.e. many different types of data, which must be handled securely). The information to be sensed and the context in which this is done changes as an automated assembly progresses. Different manipulation activities have different patterns of sensor usage. These dynamically changing sensory requirements necessitate dynamic provision of sensing services. User modules must be able to define and create sensing facilities as and when required. As noted above, the character and style of the programming environments for FAS and robots in general vary widely from rather simple procedural languages to declarative systems and, in the case of experimental systems, automatic planners. The interface offered by an SI system must correspond to the environment in which it is used.

### *3.2 Sensor Usage*

A variety of classifications of the purposes to which sensors are put in robotic systems have been proposed.[32–35] General classes of usage have, to greater or lesser extents, been supported by language constructs in commercial systems.[27,28] Work on modular sensing systems[9,10] and subsequently in the InFACT project led to the specification of a small set of sensor classes designed to accommodate the most common usages.[11,15] We now briefly consider these, giving the names used in this literature.

The most common form of sensing activity is sensing, on demand, of the **current value** of a physical state parameter of the system as a single "snap-shot". In other situations regular, periodic collection of such data is required. The data may be presented as the absolute value or relative to some previous state. The confirmation of expectations is one of the most common uses of sensor data. Expectations or error conditions can be represented in the form of predicates, defined in terms of system state parameters and threshold values and allow dynamic selection among alternative action sequences. This class is known as **state sensors**.

A common requirement is to detect a particular state of the environment. This will be done by comparing a sensor's value with a threshold (or pair of thresholds) using a relational operator. *Zforce* > 100 would be a typical example. This class of usage is known as **condition sensors** and they are used in the construction of event sensors. A condition sensor returns a boolean when interrogated.

Sensors are often used to **monitor** activities in the workspace. There are aspects that require persistent monitoring (e.g. critical conditions, which might occur at any stage of an assembly, such as excessive force or collision monitoring) and those that are relevant only at certain periods (e.g. guarded motion termination). Monitoring may be viewed in the supervisor as checking for specific occurrences, confirmation of expectations or as detection of critical conditions. This class is known as **event sensors**.

The **temporal co-ordination** of sensing activities is commonly desirable. For example, the measurement of the current co-ordinates of the robot tool may be used to compute the location of features of a part as it is swept into a binary touch probe. Monitoring of the probe can be used to trigger collection of the tool position. Nicholls and Hardy[36] discuss a number of such measurement strategies co-ordinated by binary touch probes. This class is known as **trigger sensors**.

**Reflex action** implementation may be decomposed into a condition-monitoring regime coupled to an actuation function. This is similar to the temporal co-ordination described above except that the triggered response is an actuation, rather than sensing. An important difference is that actuators tend not to be pre-emptable in the way that most sensors are (or can be made to appear). SI systems for reflex actions are not considered further here.

**Compliant motion** occurs when the motion of the manipulator is constrained by forces generated due to the task geometry. Many control strategies have been developed for this situation but, in order to achieve smooth motion at an acceptable speed, timely sensor data delivery must be guaranteed. The specification of arbitrary novel force controlled motion is noted by Lozano-Pérez to be a difficult proposition for robot programming languages.[33] Real-time sensor classes have not been included in earlier classifications. Armstrong[16] is investigating communications techniques and real-time performance in virtual sensors and is able to make timing guarantees that allow coupling appropriate to both reflex actions and compliant motion to be achieved.

On-line **quality control and inspection** procedures are increasingly integrated into the production process. By introducing quality assessment methods into assembly lines, closed loop control can be achieved, linking the appearance of a fault with the correction of it. With the growing

https://doi.org/10.1017/S0263574799001915 Published online by Cambridge University Press

complexity of production processes, Statistical Process Control (SPC) is becoming indispensable.[37] Most SPC techniques make use of logs (samples) of data. Various methods of on-line sampling and log generation may be statistically appropriate for different techniques. These methods can be supported by appropriate sensor abstractions that deliver completed logs to the supervisor. These sensors are a development of the state sensor class, with the addition of a sampling control mechanism.

## 4. SENSOR INTEGRATION SYSTEM DESIGN REQUIREMENTS

We propose two main areas of requirements: (i) flexibility and reconfigurability and (ii) integration. Each is now considered.

### *4.1 Flexibility and reconfigurability*

FAS are required to be flexible and reconfigurable at various levels of functionality. Supervisor software must reflect and manage this and the SI systems must support it. A general requirement is to handle complexity adequately. In systems where large numbers of sensors are employed, their management can be a significant task and engineering an SI system to handle them requires discipline and a good structure.

At the level of FAS configuration and maintenance, the SI system should allow the easy addition of new sensors and the replacement of existing sensors with others, perhaps based on different sensing techniques. In so far as these alterations require hardware changes and it is unlikely to be possible to make them to a running system, they represent a distinct class of changes which, though they may not be made through the SI interface by the supervisor, must be supported.

On a running system, either at significant points (such as batch to batch reconfiguration) or during the normal assembly process, it is fundamental to the overall concept that the supervisor be able to make significant reconfigurations. These will principally be the definition of new sensing functionalities based on available sensors. Dynamic reconfiguration through calibration, changing thresholds and switching between alternative data sources is also supported in this way.

### *4.2 Sensor integration issues*

Sensing hardware measures the value or change in the value of a physical world parameter, which in turn gives rise to the primary output signal, carrying the sensor specific information. The primary output may be further processed to extract required sensory parameters in a suitable form. The sensed information flows from the physical sensing end towards sensing use.

Different approaches to sensor modelling and integration have concentrated on and explored different parts of the sensor abstraction gap but it is suggested here that efforts have concentrated on the physical sensing end of the gap. The data abstraction suggestions proposed by the logical sensor approach[1] are the most prominent effort to address the issues of sensor data abstraction and integration from this side. Such efforts have covered a variable distance in the sensing abstraction gap and then any remaining processing to provide the required usage level is left to the supervisor. On the other hand, the virtual sensing approach[14] stressed concentration on the sensing usage end and explored some area near the usage end of the gap. A more detailed analysis and thorough coverage of various levels of the abstraction spectrum is required to achieve the goal of a generalised sensor integration architecture.

Researchers have suggested integration of sensing at various levels. Nnaji[30] has argued for division of sensor information into three levels for the RALPH system; (i) the data from the actual sensing process; (ii) the information after some data reduction, the level to which systems traditionally process the data and (iii) the information used for the actual application of the data. Roth and Mengel[37] argued that the term *sensor integration* is used with different connotations and can be applied at different levels of production processes. It is helpful to distinguish between: (i) integration of physical components, (ii) integration of logical functionalities, (iii) integration of sensor data on different hierarchical levels. The InFACT machine SI system supported integration at the fundamental system sensor level (see §5) and for the virtual sensor classes.

These approaches suggest that the appropriate way ahead is to divide the abstraction question into smaller self contained sensor integration problems dealing with separate areas of the abstraction spectrum. The virtual sensing approach requires that both sensor *data abstraction* and sensor *control abstraction* be considered for any division of the sensor integration problem. To develop an independent and portable sensor integration architecture the peculiarities of the specific sensing infrastructure must be handled at lower levels and hidden from the higher levels of abstract sensing. Therefore, a natural division of the sensor integration problem would be according to the increasing levels of abstraction involving both data and control. Four levels of abstraction are proposed:

1. physical sensing;
2. logical machine parameters;
3. sensor usage;
4. supervisor programming.

These levels and the sensor integration issues involved are now considered.

**4.2.1 Integration of physical sensing.** Physical sensing components include vendor specific hardware modules, hardware specific routines and device drivers. Many sensors incorporate more or less sophisticated hardware and software to condition and handle the primary signal. Self diagnostic functions for detecting and reporting problems and functions for counter-acting changes in the environment which influence the primary signal (such as change in workspace temperature or humidity) can also introduce significant extra complexity into the problem of controlling sensors. These diverse sensing devices are usually distributed across the workspace and a communications network is required.

The physical sensor level must provide a uniform

https://doi.org/10.1017/S0263574799001915 Published online by Cambridge University Press

mechanism for addressing and manipulating the hardware across the communication infrastructure and must allow the collection of raw data while maintaining efficiency. It must hide the details of device control, communication architecture, and specific software and hardware modules.

Low level sensor value and control abstraction is achieved here. All sensors are brought into a uniform type structure and a standard control mechanism is applied to them. These need not be the ones that will be used at higher levels of the system.

**4.2.2 Integration of logical machine parameter sensing.** The programming of control at the level of the state of the system measured by various physical sensors would imply knowledge of low level details about the physical sensors. This kind of dependence will result in complex code and control programs tightly coupled to the physical sensing implementation. Any replacement or enhancement of sensors would require parts of the supervisor to be rewritten. This dependence can be reduced by deciding, at a uniform level of abstraction, on the necessary base-line machine state parameters and their attributes required to measure the system for the purpose of control. These base-line requirements can be implemented through suitable physical sensors, which can be used to implement the abstract sensing functionalities. Different machine state measurement functions may use the same set of physical sensors or the sensing of a particular parameter may involve multiple and sometimes redundant sensors. Whatever physical sensing arrangements are used, they can be demonstrated to support the desired base-line abstract state sensing functions.

This level provides all the machine state parameters as fundamental system sensors (FSS) and must be implemented using the physical level (and computation). The details of implementation of the fundamental sensor level facilities and communication architecture are hidden. The association of fundamental sensors with particular devices or sets of devices is not accessible to the supervisor.

This level represents the minimal value and minimal control abstraction as it will be visible across the rest of the system. All values have one of a set of data types which provides a common framework over the rest of the architecture. A major problem with sensor integration systems is the diversity of sensor data. A well-designed strong typing system can be employed to help ensure sensor data integrity. All FSS share a single, simple usage abstraction, which allows for the return, on demand, of the current value, along with reporting of a range of possible errors. In contrast to the abstractions at the physical integration level, they are applied here to machine state parameters, not to particular devices.

**4.2.3 Integration of sensor usage abstractions.** Sensor integration at usage level is defined to combine the output of sensors to give "new" information through combined control regimes. This is the level at which virtual sensors are provided and they must be designed to be used as building blocks for more complex sensing functions. Clear and systematic mechanisms and disciplines to combine FSS and other virtual sensors to realise higher-level and more complex sensing functions are required. A central requirement is using the virtual sensing approach to analyse the sensing abstraction spectrum is therefore to develop a generalised model of a virtual sensor. Organisation and addressing of numerous abstract sensors poses a difficult problem particularly in parallel architectures and dynamic systems and must be handled here.

Through examination of common sensory requirements of robot (supervisor) programming, abstract sensing functionalities for general use can be proposed. These *virtual sensor classes* each have a well-defined interface complying with the supervisor language design.

This level therefore provides facilities for the construction and use of virtual sensors specified by the supervisor. It hides all lower parts of the SI system, only allowing the supervisor to build on FSS, and hides the details of its own implementation. Value abstraction is extended by allowing arbitrary computation and type casting. Full and flexible control abstraction is offered through the provision of both standard (see §5.3.2) and extensible (see §5.3.3) classes of virtual sensor.

**4.2.4 Integration of sensing into the supervisor programming environment.** The supervisor has to issue commands to the actuation system as well as to the sensor integration system. The sensor integration system must implement the abstract sensing behaviour while the design of sensor commands must adhere to the supervisor language paradigm. The character and style of the programming environments for FAS and robots in general vary widely. Whatever style is employed, the interface should be consistent with it and regular (or "orthogonal") across all facilities. It is particularly beneficial if error reporting is consistent across all sensing and actuation functions, allowing systematic error handling.

The sensor integration system interface can also support good engineering in supervisor design and code generation. To encourage the concepts of modularity, cohesion and low coupling, the idea of dynamic scope of the sensor functions is considered important. It means that only the currently required sensing functions are in scope and this helps to localise the sensing requirements to various sections of code and hence simplifies design and implementation of supervisor code.

An idea closely connected to dynamic scope is the sensor life-cycle concept. A virtual sensor is created with a predefined initial state; it may be used zero or more times to get sensor values through the operations it exports, and finally it is destroyed.[11] During its life, a virtual sensor can be in different modes as a result of calls to its operations and of sensor data received from the input sources. The life cycle for each class of virtual sensor with different modes, operations and functional constraints can be well defined and reflected in the interface design. This discipline supports high quality supervisor code.

The requirements for sensing functionality depend on how a supervisor programming system uses the sensory information. The main consideration can now be the role of

https://doi.org/10.1017/S0263574799001915 Published online by Cambridge University Press

sensing in programming for the application. Virtual sensors to serve the needs of the system can be designed and implemented (given that the supply of the relevant raw sensory data is in place). These might include measurement of the machine state, processing of sensed data, mechanisms for cueing and co-ordination, confirmation of expectations, satisfaction of pre-conditions, guarding actions, etc. Appropriate value and control abstraction should now be available.

*4.3 Sensor integration system design – summary*

In summary, the important aspects of the design are:

1. support for dynamic reconfiguration;
2. a general model of a virtual sensor;
3. a layered architecture providing integration at the following levels:
   (a) supervisor;
   (b) virtual sensor;
   (c) fundamental (machine state) sensor;
   (d) physical sensor.

## 5. ViSIAr – THE ARCHITECTURE

Figure 1 depicts a layered **V**irtual **S**ensor **I**tegration **A**rchitecture (ViSIAr) incorporating the considerations detailed above. It is presented as an idealised architecture offering a framework for analysis and design of real systems.

ViSIAr seeks to isolate the problems of sensing at different levels of integration and tackle them separately. The layered structure offers the normal advantage of permitting decomposition of the problem into a hierarchy of sub-problems that can be specified, analysed and dealt with independently.

The analysis presented in §4.2 suggests four major blocks: the *physical sensor system;* the *fundamental sensor system*; the *virtual sensor system* and the *supervision system*. Each system deals with a different part of the overall sensor integration problem and provides or uses specific functions with well-defined interfaces.

The supervision module is the user of the facilities exported by the virtual sensor system. It need not contain any sensor integration software but consideration of supervision language design will enforce discipline and constraints on the virtual sensing interface design to ensure supervision level integration.

ViSIAr consists of five possible layers of sensing abstraction. The two bottom layers correspond to the physical sensor system and the fundamental sensor system while the other three layers, layers 2 to 4, are the virtual sensor system. Not all five layers are directly accessible by the supervisor. Layers 0 and 1 constitute the *fixed part* of ViSIAr which is commissioned at installation time (see §4.1). The fixed part is not available to the supervisor directly. The facilities exported by the fixed part can only be accessed through the virtual sensors defined on them. On the other hand, the virtual sensor system constitutes the *dynamic part*, which enables creation of higher level sensing abstractions based on the facilities from the fixed part. The facilities defined at different levels of abstraction in the dynamic part are all equally available to the

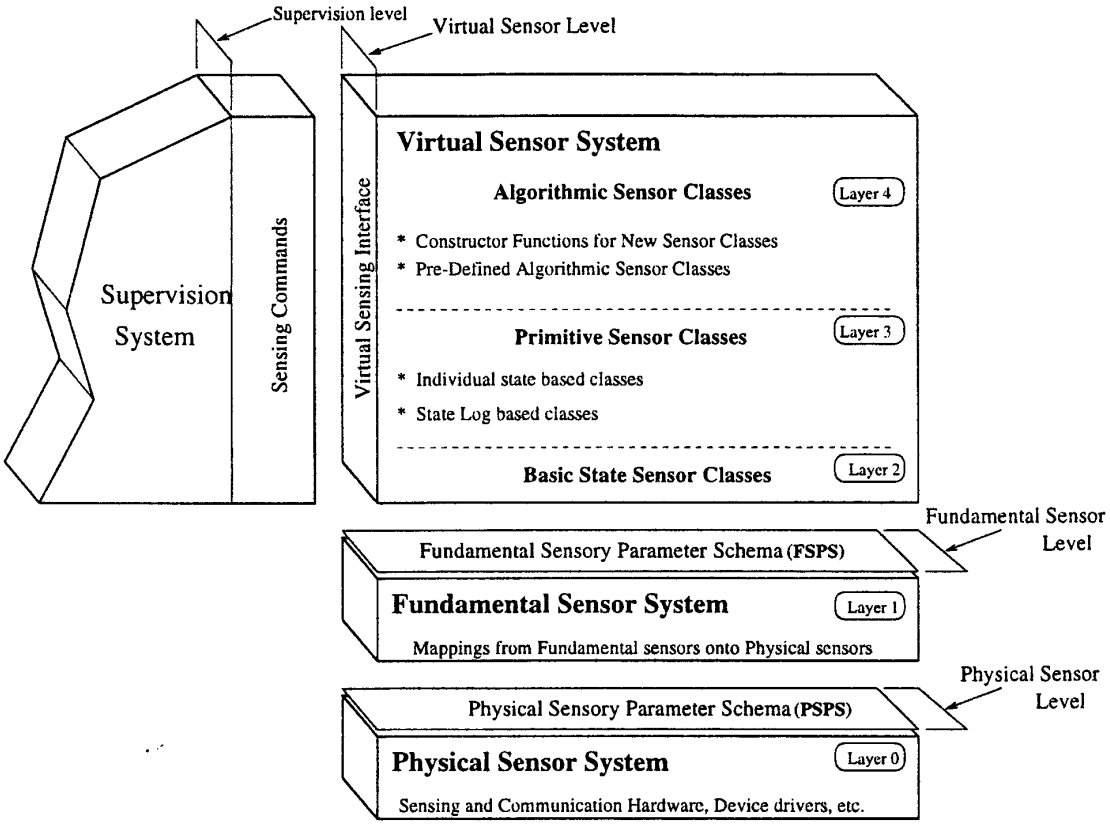



Fig. 1. Virtual Sensor Integration Architecture – ViSIAr.

https://doi.org/10.1017/S0263574799001915 Published online by Cambridge University Press

supervisor.

The fundamental and the physical sensor levels of value and control abstractions can be specified fully at design time. At the virtual sensor level only common and generic functional abstraction can be specified along with mechanisms to construct new virtual sensors with specified functions.

The specification of value and control abstraction at the interfaces of the fundamental and physical sensor levels are represented by the *fundamental sensory parameter schema* (FSPS) and the *physical sensory parameter schema* (PSPS) respectively. The virtual sensing interface is specified by the specifications of the individual virtual sensor classes. The FSPS, being a dividing line between the dynamic and fixed parts of the ViSIAr, has particular significance. It specifies the details of the total set of sensing parameters at a minimal level of abstraction to serve as a common reference for both fixed and dynamic parts. The fixed part implements the details specified in the schema and the dynamic part uses the schema details as a basis to build on. This schema also provides the dividing line to dissociate the analysis and treatment of particular sensor integration problems in the virtual and physical sensing domains.

We now consider the roles of the five layers.

### *5.1 Layer 0 – The physical sensor system*

The PSPS is the set of definitions of physical sensor parameters at the lowest level of representation. Each physical sensor has its associated entry in the PSPS. The PSPS must satisfy the sensory data requirements of the FSPS above. The details of PSPS definitions depend on the choice of the physical sensing hardware and the selection of particular physical locations where the state parameters may be sensed.

The **physical sensor system** implements the PSPS in terms of the sensory hardware and any specialised drivers. It provides operations to measure the current value of each physical sensory parameter. It may also include other operations, to reset the sensory hardware or to select a different output format for example. A design for a uniform interface has to be reached to ensure maximum independence of the integration system at higher levels. The sensory hardware devices (possibly widely distributed) are made available to the sensor data processing systems through a communication architecture.

The role of the layer is integration at the physical sensor level and it has to achieve the following two pairs of objectives:

1. to hide the vendor specific interfaces of individual physical sensors;
2. to provide a uniform interface to all physical sensors;
3. to hide the details of the communications mechanism;
4. to provide a consistent addressing and access mechanism for all physical sensors.

### *5.2 Layer 1 – The fundamental sensor system*

The FSPS is the set of definitions and descriptions of all the sensory parameters directly required to specify the physical state or changes in the state of the automated system at a minimal level of abstraction. It can also record a set of useful attributes associated with each sensory parameter, such as the resolution, units of measurement latency and rate of response. It may also specify the context of a parametric measurement, i.e., any extra knowledge of the physical environment or sensing hardware required to use or interpret the information. Examples are the location of the sensor and any relation to other sensors. During development, the FSPS can provide an important common specification for the software engineers working on higher levels of the SI system and the electronic and mechanical engineers building the physical machine.

A type system is required to support the use of data from the FSPS upwards. The nature of this is a design decision. A strong or weak, rich or limited system can be chosen as appropriate for the particular SI system implementation and supervisor. A particular sensory parameter might be modelled as a scalar or a composite data type. The sensory parameter objects may abstract one or more physical sensor's values and, conversely, a physical sensor's outputs may appear in the FSPS individually or grouped.

The **Fundamental Sensor System** implements the mappings from FSPS entries to PSPS entries. Fundamental sensors are designed to implement each of these. The set of mappings contains at least one mapping for each fundamental sensory parameter in the FSPS. Fundamental sensory parameters may have multiple map definitions that map them onto different or redundant sets of physical sensors. An automated switching mechanism between alternative mappings could be used to enhance the resilience of the system to physical sensor malfunction in a way suggested by Henderson.[1,3]

The FSPS provides the definition of sensor values being abstracted. The control abstraction for sensing at the minimal level includes at least the *Measure* operation to provide the current value of the state parameter.

This layer must achieve the following objectives:

1. to provide a fully uniform interface for current state sensing for all FSPS items;
2. to provide an addressing mechanism for fundamental sensors;
3. to hide the implementation details of FSPS to PSPS mapping functions;
4. to hide any standard low-level sensor data processing such as combining, scaling, grouping etc.;
5. to provide low-level error or malfunction detection and recovery mechanisms.[1]

In principle, a fundamental sensor can incorporate any low-level transformations of data, that is, any data processing algorithm can be defined and implemented at this level. This is, however, in the fixed part and sensors defined here are not visible and cannot be manipulated by the supervisor. Only processing known to be required in all circumstances should therefore be performed here.

Fundamental sensors provide a layer of flexibility for implementation, that is, they can be used to map the required basic sensory data to many physical sensors while at the same time the responses from these variably located

https://doi.org/10.1017/S0263574799001915 Published online by Cambridge University Press

physical sensors can be transformed, validated or combined.

*5.3 Layers 2–4 – The virtual sensor system*

The virtual sensor system provides integration of sensor usage. It supports higher level abstractions based on the facilities offered by the Fundamental Sensor System to serve the requirements of sensor based programming. Sensor data usage mechanisms are modelled and provided as virtual sensors. These high-level sensory modules are designed to abstract well-defined pieces of functionality. The interface to them must comply with the supervisory command language design and programming semantics and be designed to simplify the supervisory programming.

Virtual sensors acquire data from the fundamental sensors and implement the prescribed behaviour to provide only the required information. The data from fundamental sensors is very diverse and represents the status of different parts of the system; it usually requires further processing to be interpreted. The sensory support requirements of the user modules are diverse in terms of both data and functionality. The analysis for matching the requirements with the available facilities gives rise to varying levels of processing and complexity, and therefore varying levels of abstraction, e.g., simple measurement of a defined fundamental sensory parameter; noise reduction, combining or fusing data from various fundamental sensors; various monitoring and coordination regimes.

There are functionalities that are commonly used for simpler sensor data processing and interpretation in sensor based programming, such as testing for crossing of a given threshold. Such simple constructs may also appear as a part of complex sensor measurement and interpretation functions, for example in a logical branching strategy. This idea leads to the identification of primitive functional constructs and the design of virtual sensors to abstract them and to their use as the building blocks for more complex abstractions.

The virtual sensor system in ViSIAr has three layers of virtual sensor classes (see Figure 1) reflecting differences in the level of internal complexity. These are:

- the *basic state sensor classes* in layer 2;
- the *primitive sensor classes* in layer 3;
- the *algorithmic sensor classes* in layer 4.

The basic state sensor classes make the required fundamental system facilities available to the virtual sensor system and offer a variety of current state measurement services. The primitive sensor classes include the set of commonly required, well-defined and fixed functionalities. The algorithmic sensor classes consist of facilities to employ existing constructs and configure them in algorithms to realise complex sensor functions.

Each virtual sensor class implements the prescribed model of abstract sensor interpretation and exports it through its interface in the required format. All the virtual sensors defined in the system follow the standard virtual sensor model. The supervisor programmer can use instances of these classes by creating them dynamically.

**5.3.1 Layer 2 – Basic State Sensor Classes.** Basic state sensor classes allow the creation of virtual state sensors that interrogate the FSS. Different basic state sensor classes offer a choice of common state sensing abstractions with different semantics, e.g. single state measurement, repeated state measurement and relative state measurement.

A state sensor provides facilities, primarily, to *measure* the current value of associated sensory parameters and these are the most basic virtual sensors available to the supervisor or other virtual sensor client modules. The state sensors are the access connections to the fixed part of the sensor system for all upper layer modules.

Multiple basic sensor instances may be created simultaneously to measure the same state parameters, although all of them would be served by the same FSS. The FSS must implement adequate scheduling arrangements to serve concurrent interrogation requests from multiple state sensors.

A state sensor also *validates* the in-coming sensory data and raises exceptions that may be handled by an EDR system. Therefore, the data produced can be expected to conform to the criteria, type and format described in the FSPS.

**5.3.2 Layer 3 – Primitive Sensor Classes.** A primitive virtual sensor is a sensor module whose algorithm is predefined and fixed, and it abstracts general purpose sensing functionalities which are useful in their own right and may also be used as building blocks for higher and more complex sensing abstractions.

Condition sensors to detect specific world states; event sensors to abstract asynchronous notification of changes of state and trigger sensors to collect data when notified are important examples of classes at this level. Log sensors whose data collection is controlled in a variety of ways, which might include clock-based mechanisms, are also provided here. The derived state sensor class supports simple data processing. A derived state sensor is built using a state sensor and a function that performs the processing.

**5.3.3 Layer 4 – Algorithmic Sensor Classes.** An algorithmic sensor, as a virtual sensor, provides a control abstraction and a data abstraction. For its creation, it requires a list of virtual sensors and an algorithm to orchestrate their use.

In a programming environment where code can be manipulated, such a layer provides a means by which virtual sensors with functionalities not available among the primitive classes of layer 3 can be dynamically created. Sensors created in this way will conform to standard data and control abstractions (and therefore be usable as components in a sensor hierarchy) but their functionality can be determined at run time. The distinction between levels 3 and 4 is therefore that some algorithms have been previously recognised and provided at level 3 while level 4 offers extensibility within the framework. Derived state sensors, at level 3, are a simple model that has been recognised and can be provided.

Algorithmic sensors are examined in greater detail

https://doi.org/10.1017/S0263574799001915 Published online by Cambridge University Press

elsewhere[17] as are the possible mechanisms for building complex sensing algorithms.

### *5.4 Control abstractions and sensor classes*

Each sensor class, regardless of level, conforms to one of the standard control abstractions (see §3.2 and §4.2.3). An event sensor, for example, could be an instantiation of a standard level 3 class, probably based on the value of a state sensor crossing a threshold. Alternatively, it could be a level 4 sensor where a specialised algorithm involving a number of state sensors and some set of comparisons between their values is provided. In either case, the external view of the sensor is as an event sensor and a consumer module need not handle them differently.

It may be noted that all level 2 classes provide the state sensor control abstraction and no other.

## 6. ILLUSTRATIVE EXAMPLES

The experimental set-up consisted of a Puma 560 robot arm and a force/torque sensing wrist. The sensor integration system and supervisor were implemented on a Sun workstation under UNIX using *Ada 83*. The sensor integration system was a full working demonstration, supporting all levels of the ViSIAr architecture. Within the virtual sensor layers, a representative sample of classes was implemented. This included state sensors, derived state sensors, condition sensors, event sensors, trigger sensors, and simple algorithmic sensors. The supervisor was, in each case, only a single simple procedure that made use of SI facilities and of facilities giving access to robot control. The robot interface was implemented through the DDCMP protocol serial line giving access to the *VAL* level of control.[27] This allowed individual *VAL* commands, at both monitor and program level, to be issued. The Ada program could therefore initiate single steps or complete VAL Programs.

The sensing wrist was an FS6-120A made by the Barry Wright Corporation.[38] The output of the strain gauges built into the structure is processed by an on-board processor to deliver resolved force and torques in a re-configurable Cartesian co-ordinate system. Two-way serial communication at up to 9600 baud allows the sensor to be configured and to deliver a continuous stream of output data. A low level interface package on the Sun provided a range of procedures to configure the sensor and buffering of the output allowing collection, on demand, of the latest values.

The output from the sensor can be delivered as a 12-element record. The forces along the X, Y and Z axes and the torques about them are delivered as real values. Each of these 6 values has a re-configurable limit value. The remaining 6 output values are booleans indicating whether each has exceeded its limit. Our physical sensor is therefore "smart", reconfigurable, relatively complex and connected via serial communications using a proprietary protocol. Though reconfiguration allows control of which values it outputs, in any configuration it outputs several values continuously.

For the demonstration examples, the FSPS was defined as shown in Table I. The 6 individual real values and the 6 limit test values are provided separately. In the example code below, this will be used to advantage. To emphasise the possibility of returning composite values and the possibility of returning the same physical measurement in more than one FSPS entry, the two vectors of values are included. In a typical implementation, the 6-element force/torque record could be returned in a globally recognised data structure, which might be used with other sensing functions and with actuation.

Table I: The FSPS for the demonstration system

| Name | Data | Comment |
|---|---|---|
| Wrist-FX | scalar, real | force along the X axis |
| Wrist_FY | scalar, real | force along the Y axis |
| Wrist_FZ | scalar, real | force along the Z axis |
| Wrist_MX | scalar, real | moment about the X axis |
| Wrist_MY | scalar, real | moment about the Y axis |
| Wrist_MZ | scalar, real | moment about the Z axis |
| Wrist_State | vector, reals | the 6 values above |
| FX_Limit | scalar, boolean | FX has crossed its limit |
| FY_Limit | scalar, boolean | FY has crossed its limit |
| FZ_Limit | scalar, boolean | FZ has crossed its limit |
| MX_Limit | scalar, boolean | MX has crossed its limit |
| MY_Limit | scalar, boolean | MY has crossed its limit |
| MZ_Limit | scalar, boolean | MZ has crossed its limit |
| Wrist_Limits | vector, boolean | the 6 values above |

The PSPS specifies the data as returned by the FS6-120A interface code. The FSPS to PSPS mappings are all direct and no standard transformations are implemented at this level.

Details of the SI implementation can be found elsewhere.[17] Packages implement each virtual sensor class and the fixed part of the SI architecture. Sensors of several classes, where concurrent activity is required, are implemented as individual Ada tasks encapsulated behind simple procedure calls.

Two example supervisor procedures are now given. The first is presented as simplified Ada code and the second as pseudo-code.

### *6.1 Example 1 – A derived state sensor*

This demonstration involved the use of steel cubes with sides 1 inch (about 25mm) long. These were fixed together to form three blocks of 1, 2 and 3 cubic inches which, for each demonstration, were placed randomly at four known locations (see Figure 2). When the robot visited each

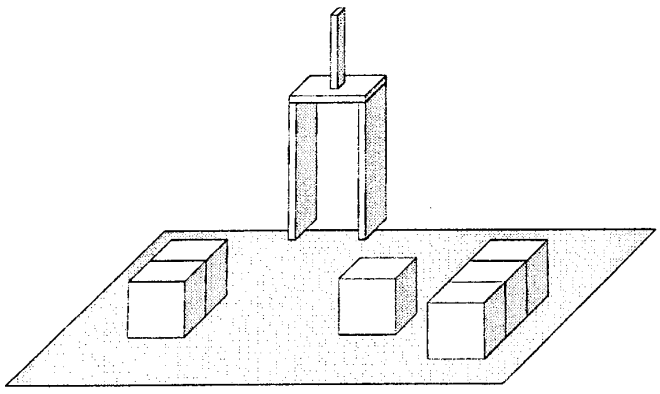


Fig. 2. Example 1 – Physical configuration.

https://doi.org/10.1017/S0263574799001915 Published online by Cambridge University Press

```
1 procedure BLOCK_COUNT is
2
3   type NUMBER_OF_CUBES is range -1..3;
4
5   ZFORCE     : SS_ID_TYPE;
6
7   BLOCK_SENS : DS_ID_TYPE;
8   CUBES      : NUMBER_OF_CUBES;
9
10  PLACES     : array (1..4) of CARTESIAN_EULER;
11                   -- Requires initialisation
12
13  ERR        : ERROR_CODE;
14
15  procedure MAP (STATE:in STATE_TYPE;
16                 OUTPUT: out NUMBER_OF_CUBES) is
17    VALUE : REAL;
18  begin
19    VALUE := VALUE_OF(STATE => STATE, COMPONENT =>1);
20    if    VALUE > -0.20 and VALUE <= 0.20  then
21      OUTPUT   := 0;
22    elsif VALUE > 1.00 and VALUE <= 1.50  then
23      OUTPUT   := 1;
24    elsif VALUE > 2.00 and VALUE <= 2.50  then
25      OUTPUT   := 2;
26    elsif VALUE > 3.00 and VALUE <= 3.60  then
27      OUTPUT   := 3;
28    else
29      OUTPUT   := -1;
30    end  if;
31  end  MAP;
32
33  package DERIVED_SENSOR is new DERIVED_STATE_SENSOR.DERIVED
34                                  (OUTYPE => NUMBER_OF_CUBES,
35                                   DERIVE => MAP);
36
37 begin
38
39  STATE_SENSOR.DEFINE(FSP    => Wrist_FZ,
40                      SS_ID  => ZFORCE,
41                      CODE   => ERR);
42  DERIVED_SENSOR.DEFINE (SS_ID => ZFORCE,
43                         DS_ID => BLOCK_SENS,
44                         CODE  => ERR);
45
46  for I in PLACES'RANGE loop
47    -- robot code to pick up block at the location PLACES(I)
48
49    DERIVED_SENSOR.INTERROGATE (DS_ID     => BLOCK_SENS,
50                                OUT_STATE => CUBES,
51                                CODE      => ERR);
52
53    PUT(" The number of cubes at place ");
54    PUT(I); PUT(" is = "); PUT(CUBES); NEW_LINE;
55
56    -- robot code to replace the block
57  end loop;
58
59  DERIVED_SENSOR.DESTROY (DS_ID => BLOCK_SENS, CODE => ERR);
60  STATE_SENSOR.DESTROY (SS_ID =>ZFORCE, CODE  => ERR);
61
62 end BLOCK_COUNT;
```

Fig. 3. Example 1 – *Ada* code.

https://doi.org/10.1017/S0263574799001915 Published online by Cambridge University Press

location in turn, grasped and withdrew, it was therefore holding a mass of 0, 1, 2 or 3 cubes. In simple application terms, a sensor was required which would indicate the number cubes held. In our set up, this had to be implemented using the Z-axis component from the force sensor (for simplicity, the blocks were picked up with the force sensor frame oriented directly downwards).

The code is shown in Figure 3. For clarity, a number of aspects of the implementation have been omitted or simplified; all packages are assumed to have been imported and their identifiers made accessible; it is assumed that IO packages for all types have been instantiated and that both the sensor integration and robot interface systems have been previously initialised and started. Error reporting in this system is implemented using error codes returned by procedure cells. These values are ignored here. Robot code and some initialisations are replaced by comments. Package names, with the "." notation are only used where they remove ambiguity or increase clarity. The Ada => notation for associating formal and actual parameters has been used as a form of commenting to improve clarity.

A derived state sensor, incorporating a function mapping from force to number of cubes (or an error) is constructed on a basic state sensor returning Z force. The sensed data is simply printed for demonstration.

To emphasise the application orientated nature of virtual sensors a special type, directly reflecting the application needs is declared on line 3. -1 is used to indicate an unknown number of cubes. A handle variable for the basic state sensor is declared on lines 5. A handle for the derived sensor and a variable to hold its returned value are then declared. PLACES, an army of robot locations, is assumed to contain the 4 sites where blocks may be placed.

The mapping function, defined on lines 15 to 31, was designed and coded empirically. A model based or learning function could be provided in a more sophisticated implementation. Derived state sensors were provided as a generic package in this *Ada* implementation. To instantiate this, a return type and a mapping function must be provided, as done on lines 33 to 35. Sensors of this class can then be created.

The body of the procedure is then conceptually simple. The state and derived state sensors are created (lines 39–44), used at each of the 4 locations (lines 46–57) and then destroyed. The de-coupling and application orientation achieved are the important points to note. Wrist_FZ is known to be an available fundamental sensor. This is declared in the FSPS and can therefore be used on line 39 without concern for the physical sensor that supports it or for any concurrent use of it or a related sensor. Within the loop, the code can be written in terms of the number of cubes. Other implementations of the "number of cubes held" sensor are clearly possible. These should not alter the use of BLOCK_SENS.

### *6.2 Example 2 – A trigger sensor*

The implementation of trigger sensors has a significant interface model component. The way in which the supervisor will be notified of the event and receive the recorded value will depend on the implementation environment. Trigger sensors have been designed for use in Occam in the InFACT system[36] and in *Ada*.[17] This example is presented as pseudo-code to provide a more general illustration. In InFACT, special purpose hardware was employed to provide guaranteed response times. Software solutions to the real-time question have been proposed.[16]

In this example, an object is carried along a defined trajectory that the designer expects to result in a collision with a flexible object. This collision is detected by the force sensor, most clearly as a Z torque. In this example, detection of the collision is used to trigger collection of the arm position. In the InFACT project, such a technique was intended to reduce uncertainties of grasp positions by a suction gripper. Pseudo-code for the module is shown in Figure 4. The sensor hierarchy is shown in Figure 5. The pseudo-code demonstrates the clarity permitted in the supervisor code by use of appropriate abstractions.

Outline Ada code for parts sorting using an event sensor in a similar configuration is given elsewhere.[31]

## 7. CONCLUSIONS

Earlier work has indicated that virtual sensors provide a supportive abstraction for robot programming. We describe an idealised framework, ViSIAr, for the design of SI systems to implement virtual sensors. A general model of a virtual sensor forms a basis for this.

Four levels of integration are proposed: physical sensor, fundamental sensor, virtual sensor and supervisor. These are justified as distinguishable requirements of an SI system and form the basis of ViSIAr. Physical sensor integration provides a uniform interface and access mechanisms for all available sensors. Fundamental sensor integration supports abstract provision of all sensory values required for control. Virtual sensor integration provides a range of sensor abstractions designed to support robot programming. It offers dynamic creation of instances of these. The virtual sensor abstractions must be made available to the robot supervisor in a way consistent with and supportive of the programming paradigm and activities found at that level. This is supervisor level integration.

```
Define state sensor P on FSS Arm_Position
Define state sensor T on FSS Wrist_Z_Torque
Define condtion sensor C on T > threshold
Define event sensor E on C
Define trigger sensor Latch on S when E occurs

    Monitor Latch
    Enable handler for trigger interrupt
    Perform activity
    If handler has been called
        use returned value for position
    else
        Abort Latch
        Error: no collision
    end if

Destroy Latch, E, C, T, P
```

Fig. 4. Example 2 – Pseudo-code.

https://doi.org/10.1017/S0263574799001915 Published online by Cambridge University Press

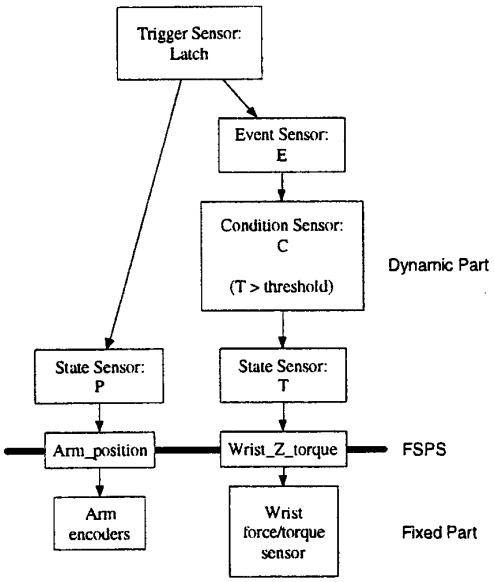



Fig. 5. Example 2 – Sensor hierarchy.

## Acknowledgements

We thank the former Institut de Cibernètica, now the Institut d'Organització i Control de Sistemes Industrials, Barcelona for provision of the DDCMP interface code and John Hinks, a postgraduate student, for force sensor interface code.

Dr. Aftab Ahmad Maroof acknowledges the financial support of the Government of Pakistan.

## References

1. T.C. Henderson and E. Shilcrat "Logical sensor systems" *J. Robotic Systems* **1**(2), 169–193 (1984).
2. Tom Henderson, "The specification of logical sensors" *Workshop on Intelligent Control* (THO146–1/86/0000/0095, © IEEE) (1985) pp. 95–101.
3. T. Henderson, C. Hansen and B. Bhanu, "The specification of distributed sensing and control" *J. Robotic Systems* **2**(4), 387–396 (1985).
4. Mohamed Dekhil and Thomas C. Henderson, "Instrumented sensor system architecture", *Int. J. Robotics Research* **17**(4), 402–417 (April, 1998).
5. G.A. Weller, F.C.A. Groen and L.O. Hertzberger, "A sensor processing model incorporating error detection and recovery" **In** (T.C. Henderson, editor) *Traditional and Non-traditional Robotic Sensors* NATO (Springer-Verlag, Berlin, 1990) pp. 351–363.
6. Michael Erdmann, "Understanding Action and Sensing by Designing Action-Based Sensors" *Int. J. Robotics Research* **14**(5), 483–509 (1995).
7. N.D. Duffy, J.T. Herd and H.J. Eccles, "Agents get framed in novel architecture" *Industrial Robot* **18**(2), 23–26 (1991).
8. N.W. Hardy, D.P. Barnes and M.H. Lee, "Declarative sensor knowledge in a robot monitoring system" **In** (U. Rembold and K. Hrmann, editors) *NATO ASI Series, 29, Languages for Sensor-Based Control in Robotics* (Springer-Verlag, Berlin, 1987), pp. 169–187.
9. J.J. Rowland and H.R. Nicholls, "A modular approach to sensor integration in robotic assembly" *Proc. INCOM '89: Symposium on Information Control Problems in Manufacturing Technology* (Madrid, Sept, IFAC (1989) **Vol 2**, pp. 473–479.
10. J.J. Rowland and H.R. Nicholls, "A modular approach to sensor integration in robotic assembly" **In** (E.A. Puente and L. Nemes, editors) *Information Control Problems in Manufacturing Technology* 1989 (IFAC, Pergamon Press, Oxford, 1989) pp. 371–376.
11. N.W. Hardy, H.R. Nicholls and J.J. Rowland, "The design of sensing commands in the InFACT assembly machine" *Proc. 23rd Int. Symp. Industrial Robots (ISIR '92)* (Nov, 1992) pp. 47–52.
12. InFACT Group, *InFACT: Project, Concepts, Machine* (MCB University Press Limited. Bradford, UK, 1992).
13. N.W. Hardy, H.R. Nicholls and J.J. Rowland, "Supervision and sensing in flexible assembly" **In** A. Morecki, G. Bianchio and K. Jaworek, editors) *RoManSy 9. Proceedings of the Ninth CISM-IFToMM Symposium on Theory and Practice of Robots and Manipulators* (Udine, Italy), Lecture Notes in Control and Information Sciences 187 (Springer-Verlag, London, 1993) pp. 385–390.
14. H.W. Hardy, J.J. Rowland and H.R. Nicholls, "Supervisory and sensing software for an integrated flexible assembly machine" **In** (M.H. Lee and J.J. Rowland, editors) *Intelligent Assembly Systems* (World Scientific Publishing Co. Pte. Ltd., 1995) pp. 101–207.
15. J.J. Rowland and H.R. Nicholls, "A virtual sensing implementation for a flexible assembly machine," *Robotica* **13**, Part 2, 195–199 (1995).
16. Edwin Armstrong, "Using virtual sensors when doing compliant motion" *Technical Report UWA-DCS-95–009* (Department of Computer Science, University of Wales, Aberystwyth, UK, April 1995).
17. Aftab Ahmad, *"A Framework for the Design of Software for Robotic Sensing" PhD thesis* (University of Wales, 1996).
18. R. Luo and M. Kay, "Data fusion and sensor integration: State-of-the-art 1990s" **In** (Abidi and Gonzales, editors) *Data Fusion in Robotics and Machine Intelligence* (Academic Press, 1992).
19. R.R. Brooks and S.S. Iyengar, *Multi-Sensor Fusion: Fundamentals and applications with software* (Prentice Hall, 1998).
20. Ronald C. Arkin, *Behaviour-based Robotics* (MIT Press, 1998).
21. R.C. Arkin and D. MacKenzie "Temporal coordination of perceptual algorithms for mobile robot navigation" *IEEE Transactions on Robotics and Automation* **10**(3), 276–286 (June, 1994).
22. R. Alami, R. Chatila, S. Fleury, M. Ghallab and F. Ingrand, "An architecture for autonomy" *Int. J. Robotics Research* **17**(4), 315–317 (April, 1998).
23. Ian Horswill, "The Polly system" **In:** (David Kortenkamp, R. Peter Bonasso and Robin Murphy, editors) *Artificial Intelligence and Mobile Robots* (AAAI Press/MIT Press, 1998) Chapter 5, pp. 125–139.
24. Sven Koenig and Reid Simmons, "Xavier: a robot navigation architecture based on partially observable markov decision process models" **In:** (David Kortenkamp, R. Peter Bonasso, and Robin Murphy, editors) *Artificial Intelligence and Mobile Robots* (AAAI Press/MIT Press, 1998) Chapter 4, pp. 91–122.
25. Joaquín L. Fernández and Reid G. Simmons, "Robust execution monitoring for navigation plans" *Proc. Intelligent Robots and Systems (IROS)*, Victoria, Canada, 1998, pp. 551–557.
26. Kurt Konolige and Karen Myers, "The Saphira architecture for autonomous mobile robots" **In:** (David Kortenkamp, R. Peter Bonasso and Robin Murphy, editors) *Artificial Intelligence and Mobile Robots* (AAAI Press/MIT Press, 1998) Chapter 9, pp. 211–242.
27. Unimation, *User's Guide to VAL-II (Programming Manual), Version 2.0 (P/N 398AG1), Part 1* (Unimation Incorporated, 200, Beta Drive, Pittsburgh, PA/5238–2987, 1986).
28. R.H. Taylor, P.D. Summers and J.M. Meyer, "AML: A Manufacturing Language" **In:** (Rathmill, editor) *Control and Programming in Advanced Manufacturing* (IFS Publications

https://doi.org/10.1017/S0263574799001915 Published online by Cambridge University Press

Ltd, UK, 1998) pp. 184–214.
29. T. Lozano-Perez, J.L. Jones and E. Mazer, *HANDEY: A Robot Task Planner* (The MIT Press, MIT, Cambridge, Massachusetts, 1992).
30. B.O. Nnaji, *Theory of Automatic Robot Assembly Programming* (Chapman & Hall, London, 1993).
31. Nigel Hardy and Aftab Ahmad "De-coupling for re-use in design and implementation using virtual sensors", *Autonomous Robots* **6**, 265–280 (1999).
32. N.W. Hardy, D.P. Barnes and M.H. Lee "Declarative sensor knowledge in a robot monitoring system" **In:** (Ulrich Rembold and Klaus Hörmann, editors) *Languages for Sensor-Based Control in Robotics* (Springer-Verlag, Berlin, 1987) pp. 169–188.
33. T. Lozano-Pérez, "Robot programming", *Proc. IEEE* **71**, 821–841 (1983).
34. C.A. Malcolm and A.P. Fothergill, "Some architectural implications of the use of sensors", **In:** (Ulrich Rembold and Klaus Hörmann, editors) *Languages for Sensor-Based Control in Robotics* (Springer-Verlag, Berlin, 1987), pp. 101–122.
35. P.J. McKerrow *Introduction to Robotics* (Addison-Wesley Publishing Company, Sydney, 1991).
36. H.R. Nicholls and N.W. Hardy, "Distributed touch sensing", **In:** (H.R. Nicholls, editor) *Advanced Tactile Sensing for Robotics* (World Scientific, Singapore, 1992) Chapter 6, pp. 107–122.
37. Narbert Roth and P. Mengel, "Sensor integration – getting the whole picture" *Sensor Review* **12**(1), 28–33 (1992).
38. Anon, *FS6–120A 6-Axis Force Sensor Software Manual* (Barry Wright Corporation, 700 Pleasant Street, Watertown, Massachusetts, 01272, 1986).

https://doi.org/10.1017/S0263574799001915 Published online by Cambridge University Press