

A Fault Tolerant Sensor Scheme¹

Tom Henderson, Esther Shilcrat and Charles Hansen

UUCS 83-003

11 November 1983

Department of Computer Science
The University of Utah
Salt Lake City, Utah 84112

Abstract

Multi-sensor systems pose the problem of how to coherently and efficiently treat the data provided by the various sensors. However, the availability of greater numbers of sensors also broadens the ability to build fault tolerant sensor systems. We define a framework in which sensors can be abstractly defined in terms of computational processes operating on the output from other sensors. Such processes are called logical sensors. Logical sensors make sensor configuration and integration easier and facilitate reconfiguration of sensor systems so that fault tolerance can be both expressed and achieved.

¹This work was supported in part by the System Development Foundation and NSF Grants ECS-8307483 and MCS-82-21750

1. Introduction

Both the availability and need for sensor systems is growing, as is the complexity in terms of the number and kind of sensors within a system. For example, most pattern recognition systems to date have been designed around a single sensor or a small number of sensors, and ad hoc configuration techniques have been used for sensor integration and operation. In the future, however, such systems must operate in a reconfigurable multi-sensor environment; for example, there may be several cameras (perhaps of different types), active range finding systems, tactile pads, and so on. In addition, a wide variety of sensing devices of different kinds, including mechanical, electronic, and chemical, are available for use in sensor systems, and a sensor system may include several kinds of sensing devices. Thus, in a multi-sensor system, the need to develop a coherent and efficient treatment of the information provided by many sensors, particularly when the sensors are of various kinds, becomes paramount.

The emergence of multi-sensor systems is one of the principal motivations for logical sensor specification. In addition, multi-sensor systems present a challenging opportunity to turn what is in one case a source of weakness (the number and variety of sensors) into a source of strength in terms of building fault tolerant sensor systems. This is the issue which we concentrate on in this paper. Other motivations include: the benefits of data abstraction and modularity, and the benefits of a hardware/software transparency so that smart sensors can easily replace software.

In single sensor systems, backup sensors would generally be duplicates of the failed sensor, or would be "functionally equivalent" to it. By "functionally equivalent" we mean that the backup sensor performs similarly to the failed sensor. However, having sensors which are to act solely as backups is not only expensive, but may also be difficult due to physical space limitations. Stopping the system in order to replace a sensor limits the effectiveness of this fault tolerance mechanism in sensor systems which are expected to run continuously or near-continuously. One answer to this problem lies in extending our view of "functionally equivalent." We concentrate on determining whether data is functionally equivalent, rather than determining if physical sensors are functionally equivalent. We take this approach to maximize the possibility of using sensors which are already doing duty in the system to produce data which is "equivalent" to that which the failed sensor would have produced. For example, the kind of data produced by a physical

laser range finder sensor could be "functionally equivalent" to that produced by two cameras and a particular stereo program. Thus, one backup for the laser range finder could be a module composed of the two cameras and the stereo program. As this example illustrates, backups may well not be simple replacement of sensors, but replacements which involve one or more sensors, and one or more software modules.

Thus, in order to take advantage of the greater opportunities for building fault tolerant sensor systems, it is necessary to express the replacement of a single sensor with a sensor-software "package" to the system. In addition, the user may need guidelines to help determine "functional equivalence." The Logical Sensor Specification Language makes use of data abstraction to build "packages" and to express fault tolerance. We explain how this is accomplished by the Logical Sensor Specification Language and describe the extensions which implement fault tolerance and which will help users design sensor systems with a greater degree of fault tolerance. We note that the inherent hardware/software transparency has been exploited as the basis for a uniform approach to fault tolerance mechanisms. We show how these extensions, together with this uniform approach, can also constitute a tool for automatic sensor system specification.

2. Logical Sensors

Logical Sensors constitute one major component of the Multi-sensor Kernel System (MKS). MKS has been proposed as an efficient and uniform mechanism for dealing with data taken from several diverse sensors [1, 2, 3, 5]. MKS has three major components: low-level data organization, high-level modeling, and logical sensor specification. The first two components of MKS concern the choice of a low-level representation of real-world phenomena and the integration of that representation into a meaningful interpretation of the real world, and have been discussed in detail elsewhere [5]. The logical sensor specification component aids the user in the (re)configuration and integration of data such that, regardless of the number and kinds of sensing devices, the data is represented consistently with regard to the low-level organization and high-level modeling techniques that are contained in MKS. However, a use for logical sensors is evident in any sensor system which is composed of several sensors or where sensor reconfiguration is desired, and the logical sensor specification component may be used independently of the other two MKS components.

Multi-sensor systems can present a user with a confusing plethora of details concerning both the sensors and associated software. However, not every detail is important in every sensor system. Logical sensors are a means by which to insulate the user from unnecessary details, and thereby allow the user to concentrate on the information which is actually necessary to determine system configuration. This is accomplished by creating "packages" of sensors, and allowing only some information about the package to be visible to the rest of the system. Going back to our earlier example, the type of data produced by the physical laser range finder sensor was also the type produced by the two cameras and the stereo program. This similarity of output result renders the alternate methods functionally equivalent, and is more important than details concerning the methods themselves. Logical sensor specification allows the user to ignore such differences of how output is produced, and treat different means of obtaining "equivalent" data as "logically" the same. We note, however, that from the fault tolerance viewpoint, type of output alone may not be enough to determine "functional equivalence" and hence a logical sensor should have visible features other than type.

A **logical sensor** is defined in terms of four parts:

1. A **logical sensor name**. This is used to uniquely identify the logical sensor.
2. A **characteristic output vector**. This is basically a vector of types which serves as a description of the output vectors that will be produced by the logical sensor. Thus, the output of a logical sensor is a set (or stream) of vectors, each of which is of the type declared by that logical sensor's characteristic output vector. The type may be any standard type (eg., real, integer), a user generated type, or a well-defined subrange of either. When an output vector is of the type declared by a characteristic output vector (i.e., the cross product of the vector element types), we say that the output vector is an "instantiation" of that characteristic output vector.
3. A **selector** whose inputs are alternate subnets (below). The role of the selector is to detect failure of an alternate and switch to a different alternate. If switching cannot be done, the selector reports failure of the logical sensor.
4. **Alternate Subnets**. This is a list of one or more alternate ways in which to obtain data with the same characteristic output vector. Hence, each alternate subnet is equivalent, with regard to type, to all other alternate subnets in the list, and can serve as backups in case of failure. Each alternate subnet in the list is itself composed of:

* A set of **input sources**. Each element of the set must either be itself a

logical sensor, or the empty set (null). Allowing null input permits **physical** sensors, which have only an associated program (the device driver), to be described as a logical sensor, thereby permitting uniformity of sensor treatment.

- * A **computation unit** over the input sources. Currently such computation units are software programs, but in the future, hardware units may also be used. In some cases, a special "do-nothing" computation-unit may be used. We refer to this unit as PASS.

A logical sensor can be viewed as a network composed of sub-networks which are themselves logical sensors. Communication within a network is controlled via the flow of data from one sub-network to another. Hence, such networks are data flow networks.

3. Fault Tolerance

The Logical Sensor Specification Language has been designed in accordance with the view that languages should facilitate error determination and recovery. As we have explained, a logical sensor has a selector which takes possibly many alternate subnets as input. The selector determines errors, and attempts recovery via switching to an another alternate subnet. Each alternate subnet is an input source - computation unit pair. Selectors can detect failures which arise from either an input source or the computation unit. Thus, the selector together with the alternate subnets constitute a failure and substitution device, that is, a fault-tolerance mechanism, and both hardware and software fault tolerance can be achieved. This is particularly desirable in light of the fact that "fault tolerance does not necessarily require diagnosing the cause of the fault or even deciding whether it arises from the hardware or software" (emphasis added) [4]. In a multi-sensor system, particularly where continuous operation is expected, trying to determine and correct the exact source of a failure may be prohibitively time-consuming.

Substitution choices may be based on either replication or replacement. Replication means that exact duplicates of the failed component have been specified as alternate subnets. In replacement a different unit is substituted. Replacement of software modules has long been recognized as necessary for software fault-tolerance, with the hope, as Randall states, that using a software module of independent design will facilitate coping "with the circumstances that caused the main component to fail" [4]. We feel that replacement of physical sensors should be exploited both with Randall's point in view, and because extraneous considerations, such as cost, and spatial limitations as to

placement ability are very likely to limit the number of purely back-up physical sensors which can be involved in a sensor system.

3.1. Recovery Blocks

The recovery block is a means of implementing software fault tolerance [4]. A recovery block contains a series of alternates which are to be executed in the order listed. Thus, the first in the series of alternates is the primary alternate. An acceptance test is used to ensure that the output produced by an alternate is correct or acceptable. First the primary alternate is executed, and its output scrutinized via the acceptance test. If it passes, that block is exited, otherwise the next alternate is tried, and so on. If no alternate passes, control switches to a new recovery block if one (on the same or higher level) is available; otherwise, an error results.

Similarly, a selector tries, in turn, each alternate subnet in the list, and tests each one's output via an acceptance test. However, while Randall's scheme requires the use of complicated error recovery mechanisms (restoring the state, and so on), the use of a data-flow model makes error-recovery relatively easy. Furthermore, our user interface computes the dependency relation between logical sensors [1]. This permits the system to know which other sensors are possibly affected.

The general difficulties relating to software acceptance tests, such as how to devise them, how to make them simpler than the software module being tested, and so on, remain. It is our view that some acceptance tests will have to be designed by the user, and that our goal is simply to accommodate the use of the test. Unlike Randall, we envision the recovery block as a means for both hardware and software fault-tolerance, and hence we also allow the user to specify general hardware acceptance tests. Such tests may be based, for example, on data link control information, 2-way handshaking and other protocols. It is important to note that a selector must be specified even if there is only one subnet in a logical sensor's list of alternate subnets. Without at least the minimal acceptance test of a "time-out," a logical sensor could be placed on hold forever even when alternate ways to obtain the necessary data could have been executed. Given the minimal acceptance test, the selector will at least be able to signal failure to a higher level selector which may then institute a recovery. However, we also wish to devise special schemes for acceptance tests when the basis for substitution is replacement.

While users will often know which logical sensors are functionally equivalent, it is also likely that not all possible substitutions of logical sensors will be considered. Thus, we are interested in helping the user expand what is considered functionally equivalent. Such a tool could also be used to automatically generate logical sensors. We give an example logical sensor network in Figure 1. This example shows how to obtain surface point data from possible alternate methods. The characteristic output vector of Range_Finder is $(x:real, y:real, z:real)$ and is produced by selecting one of the two alternate subnets and "projecting" the first three elements of their characteristic output vectors. The preferred subnet is composed of the logical sensor Image_Range. This logical sensor has two alternate subnets which both have the dummy computational unit PASS. PASS does not effect the type of the logical sensor. These alternatives will be selected in turn to produce the characteristic output vector $(x:real, y:real, z:real, i:integer)$. If both alternates fail (whether due to hardware or software), the Image_Range sensor has failed. The Range_Finder then selects the second subnet to obtain the $(x:real, y:real, z:real)$ information from the Tactile_Range's characteristic output vector. If the Tactile_Range subsequently fails, then the Range_Finder fails. Each subnet uses this mechanism to provide fault tolerance.

3.2. Ramifications of Fault-Tolerance Based on a Replacement Scheme

Many difficult issues arise when fault tolerance is based on a replacement scheme. Because the replacement scheme is implemented through the use of alternate subnets, the user can be sure that the type of output will remain constant, regardless of the particular source subnet. Ideally, however, we consider that a replacement based scheme is truly fault tolerant only if the effect of the replacement is within allowable limits, where the allowable limits are determined by the user. As a simplified example, consider a sensor system of one camera, A, and a back-up camera, of another type, B. Suppose camera A has accuracy of $\pm 0.01\%$, and camera B has accuracy of $\pm 0.04\%$. If the user has determined that the allowable limit on accuracy is $\pm 0.03\%$, then replacement of camera A by camera B will not yield what we call a truly fault tolerant system; if the allowable limit is $\pm 0.05\%$, the replacement does yield a truly fault tolerant system, as it will if the user has determined that the system should run regardless of the degree of accuracy.

As mentioned above, determining functional equivalence may necessitate seeing more of a logical sensor than merely its type. This example illustrates this point in that we

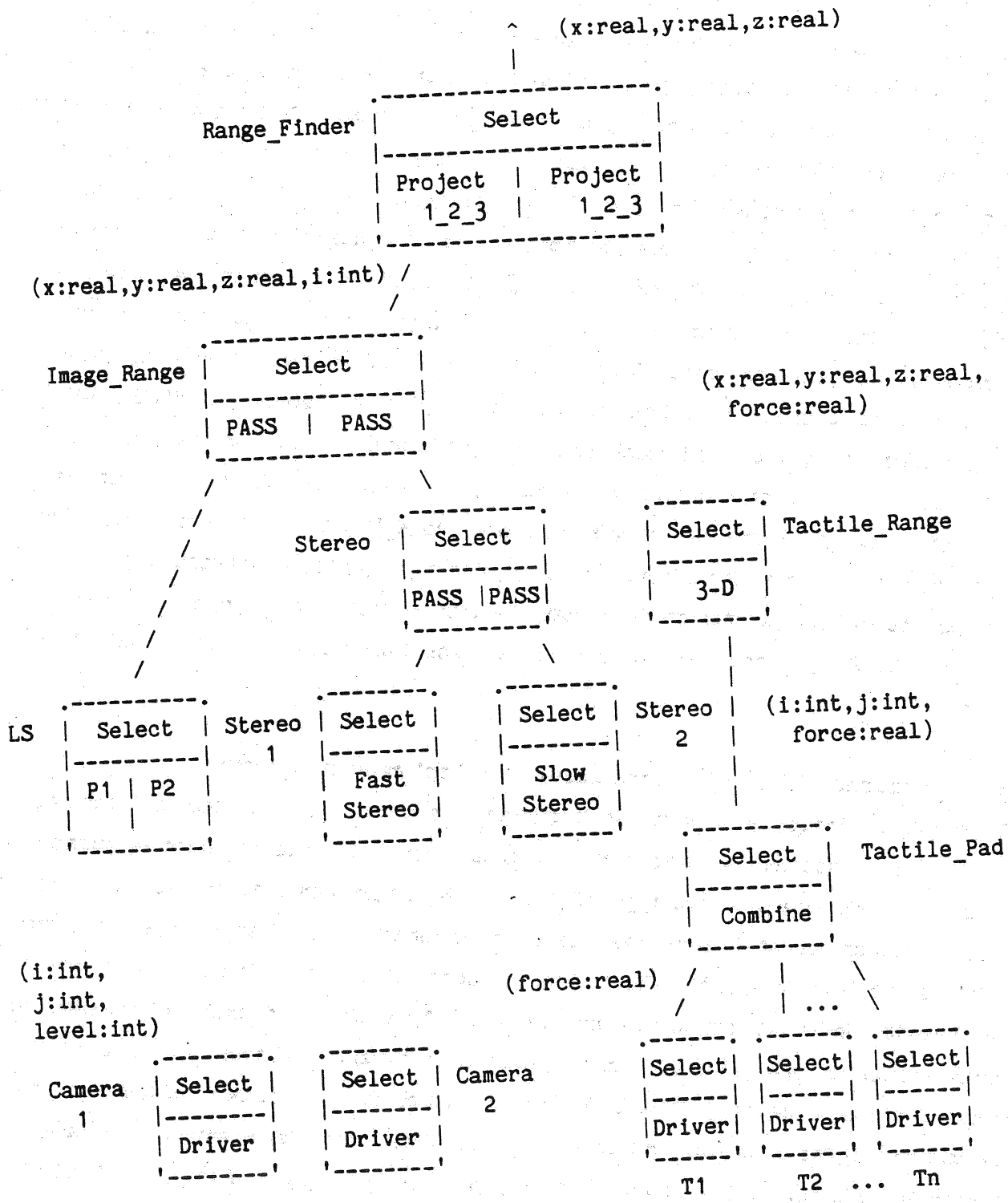


FIGURE 1: Logical sensor network for Range_Finder.

have isolated a need to know more about leaf logical sensors (physical sensors). However, we also mentioned that the above example was simplified. Let us now assume, in addition, that the user can use a variety of algorithms to obtain the desired final output. Suppose one of those algorithms incorporates interpolation techniques which could increase the degree of accuracy over camera B's input. In this case, the user may be able to use camera B and this algorithm as an alternate subnet and have a truly fault tolerant system, even if camera B's output is not itself within the allowable accuracy limit. Thus, when we consider a slightly more complex example, we see a general need for having features (beside type of output) of logical sensors visible, and a need to propagate such information through the system.

Feature propagation, together with allowable limit information, is needed for replacement based fault-tolerance schemes, and constitutes an acceptance test mechanism. In addition, such feature propagation has a good potential for use in automatic logical sensor system specification/optimization. For example, consider a work station with several sensors. Once various logical sensors have been defined and stored, feature propagation can be used to configure new logical sensor with properties in specified ranges, or to determine the "best" (within the specified, perhaps weighted, parameters) logical sensor system. Thus, feature propagation is necessary for both fault tolerance and automatic generation of logical sensor systems, and it is our view that the basic scheme will be the same in either case.

4. LSS: An Implementation of Logical Sensors

A Logical Sensor Specification system, LSS, has been developed and implemented in the "C" programming language under UNIX, a registered trademark of Bell Labs. This specification system provides a user-interface for interactively editing sensor systems, networks. This system allows the capability of providing alternate subnets for assisting the fault-tolerance issue as well as computing the dependency relation between sensors as previously mentioned.

When entering a new system, the user begins by building logical sensors based on the physical sensors available. These sensors may be used to construct other logical sensors consisting of input vectors from other logical sensors, a computational unit, and a characteristic output vector. The user may specify alternate subnets to be selected in

case of sensor failure by giving a computational unit and its various input vectors. All alternate subnets for a particular sensor produce the same characteristic output vector. The system facilitates interactive editing of sensor systems by allowing the user to either delete or modify a particular logical sensor thus modifying subnets. If this alteration affects any other sensors in the network, the user is notified of this problem.

To exemplify this, we present portions of the specification of the sensor system given in Figure 1. Figure 2 shows the screen layout after the declaration of the physical device, camera_1, used in the logical sensors: LS, Fast_Stereo, and Slow_Stereo. This logical sensor has no inputs since it is a physical device and is characterized by the output vector (x:real,y:real,i:integer). The computational unit is the device driver for the camera. There are no alternate subnets specified for this logical sensor.

LOGICAL SENSOR NAME: camera_1

| ALTERNATE SUBNET 1 | | PROGRAM ID: camera_driver | |
|--------------------|--------------|------------------------------|--------------|
| INPUT SENSOR NAME | INPUT VECTOR | CHARACTERISTIC OUTPUT VECTOR | |
| | | ELEMENT NAME | ELEMENT TYPE |
| | | x | real |
| | | y | real |
| | | i | integer |

```
enter command> input camera_1
Another Subnet? [y or n] n
enter command>
```

FIGURE 2: CRT screen after specifying camera_1

Figure 3 demonstrates the specification of Fast_Stereo. This logical sensor is specified in terms of two input vectors, from camera_1 and camera_2, the computational unit "fast_stereo" and the characteristic output vector, (x:real,y:real,z:real,i:integer). This logical sensor has no alternate subtrees defined. The asterisks by the input vectors indicate which elements are to be used by the computational unit.

LOGICAL SENSOR NAME: Stereo_1

| ALTERNATE SUBNET 1 | | PROGRAM ID: fast_stereo | |
|--------------------|--------------|------------------------------|--------------|
| INPUT SENSOR NAME | INPUT VECTOR | CHARACTERISTIC OUTPUT VECTOR | |
| | | ELEMENT NAME | ELEMENT TYPE |
| camera_1 | * x | x | real |
| | * y | y | real |
| | * i | z | real |
| | | i | integer |
| camera_2 | * x | | |
| | * y | | |
| | * i | | |

```
enter command> input Stereo_1
Type '*' to Select input or space to skip
Another Subnet? [y or n] n
enter command>
```

FIGURE 3: CRT screen after specifying Stereo_1

Next we'll show the specification for the top level logical sensor Range_Finder. This logical sensor has the characteristic output vector (x:real,y:real,z:real) and is composed of two alternate subnets. The first alternate subnet is shown in Figure 4. This subnet is composed of the input vector, (x:real,y:real,z:real,i:integer), from the logical sensor Image_Range and the computational unit "Project 1_2_3" which will project the first three elements of its input vector.

LOGICAL SENSOR NAME: Range_Finder

| ALTERNATE SUBNET 1 | | PROGRAM ID: Project_1_2_3 | |
|--------------------|--------------|------------------------------|--------------|
| INPUT SENSOR NAME | INPUT VECTOR | CHARACTERISTIC OUTPUT VECTOR | |
| | | ELEMENT NAME | ELEMENT TYPE |
| image_range | * x | x | real |
| | * y | y | real |
| | * z | z | real |
| | i | | |

```
enter command> input Range_Finder
Type '*' to Select input or space to skip
Another Subnet? [y or n] y
enter command>
```

FIGURE 4: CRT screen after specifying first subnet of Range_Finder

Figure 5 shows the specification of the second alternate subtree which consists of the input vector, (x:real,y:real,z:real,f:real), from the logical sensor Tactile_Range and the same computational unit as the first alternate subnet, "Project 1_2_3". The remaining portions of the entire network were defined in a similar way.

LOGICAL SENSOR NAME: Range_Finder

| ALTERNATE SUBNET 2 | | PROGRAM ID: Project_1_2_3 | |
|--------------------|--------------|------------------------------|--------------|
| INPUT SENSOR NAME | INPUT VECTOR | CHARACTERISTIC OUTPUT VECTOR | |
| | | ELEMENT NAME | ELEMENT TYPE |
| tactile_range | * x | x | real |
| | * y | y | real |
| | * z | z | real |
| | f | | |

```
enter command> input Range_Finder
Type '*' to Select input or space to skip
Another Subnet? [y or n] n
enter command>
```

FIGURE 5: CRT screen after specifying Range_Finder

5. Current Research Issues

We are currently investigating several aspects of logical sensor systems:

- * Semantics of Logical Sensor Systems. Both the operational and denotational semantics of logical sensor systems require thorough investigation if the fundamental properties of logical sensor systems are to be understood.
- * Sensor/Algorithm Performance Evaluation. It is crucial in many applications to know the effect of passing data of known characteristics through some algorithm implemented on a certain architecture. For, example, if an algorithm

merges data from two different resolutions, its output most probably is of the lower resolution of the two. On the other hand, some algorithms actually improve the quality of the data (e.g., subpixel feature detectors in images).

- * Automatic Logical Sensor Generation. Given an expert system on sensors and algorithms which work on those sensors, it may be possible for an AI system to demand new logical sensors based on the kinds of objects or features that it needs to detect in the world. The simplest example would be new sensors with constants instead of variables for some element of the characteristic output vector. E.g., given a logical sensor which detects circles of any radius, a logical sensor could be easily generated to detect circles of a fixed radius.
- * Implementation Issues. Finally, there are the issues of efficiency and robustness which must be addressed. It is imperative to provide a system which performs in real-time and with low probability of unrecoverable error. Even the characterization of the probability of error is difficult.

References

- [1] Hansen, C., T.C. Henderson, Esther Shilcrat and Wu So Fai.
Logical Sensor Specification.
In Proceedings of SPIE Conference on Intelligent Robots, pages 578-583. SPIE,
November, 1983.
- [2] Henderson, Thomas C. and Wu So Fai.
A Multi-sensor Integration and Data Acquisition System.
In Proceedings of the IEEE Conference on Computer Vision and Pattern
Recognition, pages 274-280. IEEE, June, 1983.
- [3] Henderson, T.C. and Wu So Fai.
Pattern Recognition in a Multi-sensor Environment.
UUCS 001, University of Utah, July, 1983.
- [4] Randell, B.
System Structure for Software Fault Tolerance.
Prentice-Hall, Englewood Cliffs, NJ, 1977, pages 195-219.
- [5] Wu So Fai.
A Multi-sensor Integration and Data Acquisition System.
Master's thesis, University of Utah, June, 1983.

