Sensors and Actuators A, *32* (1992) 491–498

Data structures for multi-sensor integration

A. Ruiz, D. Guinea, L. J. Barrios, P. Bustos and F. Betancourt

Instituto de Automática Industrial (CSIC), CN III, Km 22,800 La Poveda Arganda del Rey, 28500 Madrid (Spain)

Abstract

This paper analyzes data organization requirements for multi-sensor information processing. It proposes solutions for both abstract data structures and storage formats. We formalize the concept of general sensor information and define a data structure able to support sensor information obtained by a wide spectrum of applications. The problem of off-line storage is studied and two different solutions are proposed. Finally, we comment on some guidelines for designing real-time applications under our approach.

1. Introduction

There is a marked contrast between the latest smart sensors, which are able to measure a broad set of physical magnitudes, and the actual possibilities of extraction of meaning and interpretation of the data acquired by them to carry out intelligent and complex tasks [1, 2]. Many current sensor applications may be expressed in terms of the 'artificial perception' paradigm: to infer the state of the system (working universe) from the sensor data.

This paper presents some ideas on data structures and organization in order to implement an efficient scheme for multi-sensor information integration.

The state of the system

Given a certain system, we use a set of sensors to monitor relevant physical or chemical magnitudes, expected to be related to states of interest. These states may correspond to control decisions over the system:

Binary decision. There exist only two possible states in the working universe: 'Yes—No', 'Correct working—Faulty', etc.

Classification problems. There is a finite set of disjoint options for the system: phoneme recognition, character recognition, object identification, detection of impact regions, etc.

Estimation of continuous variables. In this case we need to determine the value of a certain parameter or magnitude. It can be seen as a classification problem with infinite states (a continuum of classes) e.g., tool wear in machine tools [3], detection of objects position, etc.

Arbitrary patterns generation. The most general result which can be obtained by sensor information processing is a set of numbers with a certain structure, as complex as needed: signal filtering, image enhancement, associative memories, etc.

Many practical applications are classification problems (finite number of states of interest). Continuous magnitudes estimation may be handled using this approach after a discretization process. The perceived state will be normally used in a following planning stage of control actions. In simple applications it is possible to associate directly each state with a control action (no planning required).

Sensor information

Let $\mathbf{f}(\mathbf{x},t)$ be the value of some physical magnitude at time t, in a space position $\mathbf{x}$. In general this magnitude will not be directly measurable, so we have to use *sensors* or *transducers*. Therefore we acquire $\mathbf{m}(\mathbf{x},t)$ somewhat related with $\mathbf{f}$. If the transfer function of the transducer is known, it is possible to calculate $\mathbf{f}$ from $\mathbf{m}$. In any case, the perception objective must be reached by processing the directly available measure $\mathbf{m}(\mathbf{x},t)$. This may cause several problems: data consistency between different sensors, calibration, etc. There

0924-4247/92/$5.00

may be also time dependencies in sensor response: drifts, change of climatic conditions, disfunctions, etc. These difficulties could be treated including all changing factors in the sensor transfer function, calibrating the system before data acquisition or even ignoring them when they are not very serious.

We consider a *sensor* as a function $\mathbf{m}(\mathbf{x},t)$ which obtains the value of some physical variable at the position $(\mathbf{x},t)$. Without lack of generality, we will only consider scalar sensors. Each component of a vectorial sensor may be regarded as an independent scalar sensor. Sensor data need to be processed by digital devices, so m values must be discretized into a finite number of levels. Also, the 'sensor function' $\mathbf{m}(\mathbf{x},t)$ will be defined only for specific time instants, usually periodic ($t = nT$), according to a constant *sampling rate.*

On the other hand, let $C_i(t)$ be the region of the physical space where the sensor $m_i(\mathbf{x},t)$ has a well-defined value. $C_i(t)$ is the *directly observable region* for the ith sensor. Only after some processing, it should be possible to obtain information related to remote regions. If $C_i(t)$ consists of just one point in the space, m_i is a *point sensor* (temperature or pressure sensors, microphones, etc.). If $C_i(t)$ is a finite set of points, $m_i(t)$ may be referred to as a *structured sensor.* Depending on spatial ordering of $C_i(t)$ there are array sensors with one, two, three dimensions, etc. (video cameras, artificial skins, etc.). We don't have to consider an infinite number of points in $C_i(t)$ because it is impossible to perform an infinite number of measures. Any structured sensor may be divided into a number of point sensors. If $C_i(t)$ does not depend on the time t, m_i will be a *static sensor.*

Any arbitrary set of sensors can be redefined as a set of point scalar sensors $m_i(\mathbf{x},t)$ for $i = 1 \ldots N$. The information generated by them is a sequence of elemental observations with the general form: [time, position, sensor, measure]: $[\mathbf{t}\ \mathbf{x}\ \mathbf{i}\ m_i(\mathbf{x},t)]$

$$I = (\ldots, [t_1 \mathbf{x}_1\, i\, m_i(\mathbf{x}_1,t_1)], [t_2 \mathbf{x}_2\, j\, m_j(\mathbf{x}_2,t_2)], \ldots)$$

which includes all the information acquired by the multi-sensor system. Unfortunately it is not easy to design processing tools for so general a data structure. It is sensible to allow some simplifications.

2. Towards a practical data structure

When the space coordinates of the sensors do not depend explicitly on time (e.g., static sensors with constant coordinates) it is not necessary to include them in the sequence I, being redundant information.

$$I = (\ldots [t_1\, i\, m_i(t_1)], [t_2\, j\, m_j(t_2)] \ldots)$$

Assuming we have the measures of all sensors at time instant t:

$$I = (\ldots [t\, m_1(t) m_2(t) \ldots m_N(t)] \ldots)$$

Finally, when the measure instants are known (i.e., periodic), it is possible to remove the time in the components of I:

$$I = (\ldots [m_1 m_2 \ldots m_N], [m_1 m_2 \ldots m_N] \ldots)$$

Under the above-mentioned assumptions, which hold for a set of synchronized static sensors, their data may be represented by a sequence of vectorial observations. Space and time locations are not relevant, so they do not appear explicitly. Those vectorial observations may be considered as typical *patterns* as used by classical pattern recognition techniques. In particular, they can be considered as observations of a multi-dimensional random variable, its probability density function depending on the state of the working universe.

Several artificial perception applications may be handled with the previous model. However, in most cases practical difficulties appear:

mobile sensors: changes in space coordinates of observations complicate data interpretation tasks;

timing: if sensors generate data at arbitrary times (e.g., different sampling rates, specific triggers, random events, etc.) it is not always possible to obtain, at a certain time t, complete observation patterns (not all components are defined at the same time);

noise: even if we could always get complete patterns, the effect of noise (present in every real system), decreases reliability of a single observation;

sensor dynamics: real sensors usually show a 'time evolution'. Their responses to a specific stimulus have a certain duration, so a unique measure will not always be enough to determine the state of the system. We should take into

account their responses during a period of time. It is important to decide whether this period length must be constant or variable, whether successive measures must overlap or not, etc.

These problems arise from the requirement of an easy data processing and from acquisition constraints.

In the following we will assume that sensor data will be processed by *examples based* pattern classifiers, appropriated for a wide spectrum of recognition problems. These techniques require a set of examples representative enough (in a statistical sense) of the states to be recognized. This set can be more easily built provided there are a common structure and acquisition conditions for the sensor examples used in training, testing and real-time execution. Therefore, these techniques are convenient for static sensors with periodic sampling times. For example, working with mobile sensors may require a training set too high to cover all the relevant situations.

Timing problems, noise and sensor dynamics may be handled in a unified way, defining an elemental piece of sensor information as the set of successive observations (time series) obtained, in such a way that they constitute a repetitive entity. We can group data corresponding to a time interval which is a common multiple of all acquisition periods. The sampling rate for a signal must be, at least, twice its fastest frequency, and the total length of the time series acquired is related to the lowest frequency of interest. Hence, sensor information is structured in a sequence of elemental packets, each of them associated to an initial time instant and carrying the sensor responses generated during a specified time interval.

The proposed patterns structure could be processed by a broad set of techniques. However, many acquisition systems include both periodic and asynchronous variables, triggered by certain events in the working universe. Integrating sensors with timing problems may be managed using different approaches. For instance, we could consider separately synchronous and asynchronous signals making an independent recognition stage and fusing results at a higher level. Another option may be the generation of complete patterns, assigning the most safe data (obtained in preceding times) to the missing values. Both solutions can be easily supported by the proposed pattern structure.

Data inconsistency, irregular data structures, lack of normalized conditions, are basic problems in sensor data integration and interpretation. Frequently, pattern recognition is a mathematically solved subject, provided there exists a statistically valid training set of patterns. Essentially, it is a question of learning time and a compromise between response time and success rate.

The most used and useful recognition techniques are supervised: it is necessary to provide the class or state the patterns belong to. As we saw in the preceding Section, this state may be specified with different degrees of complexity, depending on the nature of the recognition problem. The sensor data structure should be flexible enough to allow different alternatives for class specification.

On the other hand, fast sampling rates imply too-large elementary data packets for classical recognition tools. This suggests a previous stage, usually called *feature extraction*, to reduce the amount of raw data. This is an open research line, not very satisfactorily solved nowadays except in very simple situations. The idea is to find a transformation from the pattern space to a new, low dimensional feature space in which overlapping between classes does not increase too much. Therefore, a good sensor data structure must be able to allocate both original and transformed data.

'Data' should be separated from 'processing commands' in order to allow different perceptual (classification) objectives from the same original data. Evaluation of alternative processing schemes is only possible if data are not *biased* towards particular processing approaches. For instance, it is not satisfactory to specify in the data structure parameters as iteration numbers, metrics to be used, etc., required by some classification tools.

Finally, it is reasonable to include significative acquisition and environmental conditions in the data structure as well as remarks and commentaries provided by the user for documentation purposes.

Following previous requirements we propose a particular sensor data structure. It has been used, with minor modifications, in refs. 3 and 4.

The sequence of recepts

The whole data structure employed to support arbitrary sensor information through all processing stages is called 'sequence of recepts'. We take the term *recept* from Sobolewski in ref. 5.

A recept is the elementary sensor observation or example, i.e., the data set generated by the sensors during a time interval at a specific position. It is a substructure located in a memory region shared by acquisition devices and processing programs. This structure is updated and modified by application-specific acquisition drivers, feature extraction programs and recognition tools.

The 'recept structure' contains the following fields (Fig. 1):

(1) Structure

—*Number* of sensors (N);

—*Identifiers* of each of them (type string);

—*Amount* of data acquired by each sensor (possibly multi-dimensional: 256, 64×64, $8 \times 8 \times 3$, etc.);

(2) Data addressing

—*Acquisition coordinates* (for each sensor s: x_s, y_s, z_s);

—*Initial time* of acquisition (t);

—*Acquisition conditions* for each sensor: offset, amplifying factor, sampling rate (Of_s, Sc_s, dT_s)

—*Data read:* the array $x(s,i)$ contains the ith measure (binary counts in the AD converter) of the sth sensor in the current recept, corresponding to time $= t + i\,dt(s)$. The physical magnitude value m is computed easily: $m = (x(s,i) - Of_s)Sc_s$. Multidimensional sensors may be allocated as one-dimensional vectors by proper indices transformation;

—*External conditions* which depend on the concrete sensorial system. We assume that the recept contains a list of attributes with its values (numeric or symbolic). For instance: 'speed' = 18.5; 'material' = B3, etc. If these values vary very frequently, they could be considered as conventional sensors.

(3) Learning

—*Recept type:* there are two options: 'teacher' and 'unknown'. The former includes the class it belongs to, so it can be used in supervised learning stages, for training or testing. The latter is not labelled with any class. It can be used only in real work of the system or for non-supervised learning (cluster techniques).

—*Class code:* this field specifies how the class or state is coded (for 'teacher' recepts): by means of integer numbers, real vectors, binary vectors, symbols, etc. Sometimes it is necessary to include the code type (BCD, positional, etc.) in order to interpret binary vectors. Finally, it is reasonable to specify complementary information as the maximum number of classes, length of the binary code, etc.

—*Class* (in 'teacher' recepts): to designate the class of the recept, according to the class code field. A probability factor could be included to express the confidence in that classification for the recept. Another option is to maintain a list of the probabilities of all classes (a priori), which can be updated (to become a posteriori) by the recognition tools.

(4) Miscellaneous

—*Documentation:* literal information provided by the user (one or more strings).

—*User defined:* it is reasonable to allow user-defined fields to contain significative information specific to the current application.

Those fields constitute the 'recept structure'. It should be noted that only a part of them changes during system operation, in particular the data array $x(s, i)$ and the class of the recept, among others. Certain fields (defining the recept template: number of sensors, data length of them, the class codification selected, etc.) are set up at system initialization.

Now we describe the operations of the whole structure (Fig. 2), 'the sequence of recepts structure'.

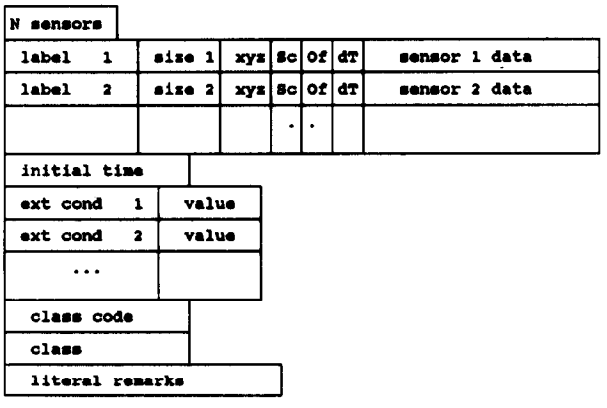


Fig. 1. The recept structure.

(1) Sequence opening: we must provide two identifiers.

—*Sequence name:* to specify which real sensor system is selected. If that name does not refer to any real system, we assume that the sequence of recepts is stored in a disk file. Initialization tasks of acquisition drivers are executed.

—*Features extraction program filename* (or command string): to perform a transparent processing stage over the recepts adapting them to the needs of the program that opens the sequence (i.e., a pattern classifier). If this identifier is the null string (''), no transformation of the recepts will be done.

(2) Get new recept: to ask the real acquisition driver or the disk file for the latest recept generated by the sensors. When the new recept is available the RECEPT structure is updated and transformed according to the features extraction program specified in the sequence opening operation. Therefore any program may use, automatically, a vector of features instead of big amounts of raw data.

(3) Sequence close: to free resources, close files, etc.

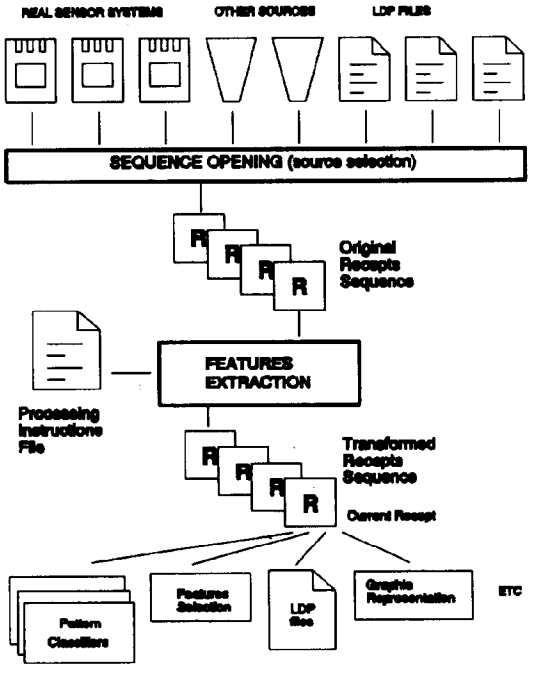


Fig. 2. Sequence of recepts operations.

3. Storage formats

Working the sequences of recepts obtained in real time from real sensors is not an easy task. It is necessary to use specialized architectures, digital signal processors, etc. However, implementation of the previous operations is easier when we restrict our attention to stored data files. We cannot lose data, no matter how long is the processing time duration. Therefore it is possible to compare and evaluate in a conventional computer (off line) different processing strategies to solve each recognition task.

It seems necessary to define a disk file format to store sequences of recepts. The basic problem in the analysis of different options is to avoid storing redundant information in order to minimize file sizes and optimize data addressing.

A first solution is to use files whose base type is the same recept structure (probably a record). Recepts access time is short but, unfortunately, constant array length of some programming languages forces an over-sized structure, wasting disk space.

Another option is to define a formal language or format for ASCII files with key words, headlines and the like, in which we only have to include relevant information (changing from one recept to another). This solution presents low storage optimization and slow (sequential) access to recepts. However, files are legible and easily modifiable. Figure 3 shows the formal storage language LDP (perception data language) used in several applications by our research group, as well as a short example file written in this format (Fig. 4).

We could alternatively use multiple files, a set-up file with format specifiers as sizes, codes, and so on, common for the whole sequence, which is logically attached to one or more data files (integer type for raw data and real type for features) which optimally store the information. It presents just a small organization problem, and it is very useful for applications with very large amounts of data. However, files are not easily editable.

Although the advantages of a common structure for both raw and processed data is clear, we must notice an important fact: a great deal of data processing is made over vectors of features. We only work with original sensor data at first stages.

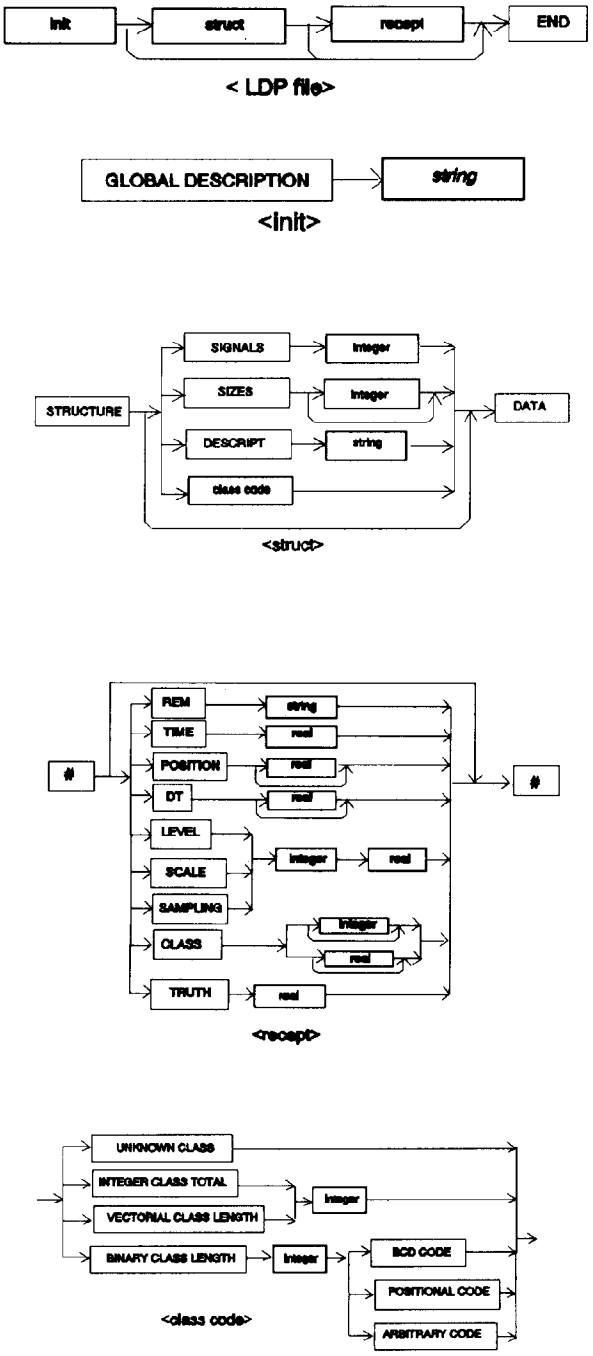


Fig. 3. LDP syntax.

Therefore, separating raw and processed data files provides generality to the sensor systems without loss of access efficiency to small vectors of features.

```
GLOBAL DESCRIPTION
  "Impacts over an artificial skin. Impact energy = 20g·30cm·g
   Accquistion circuit. Master examples over all 7 skin cells.

STRUCT

  SIGNALS 1
  SIZES 200
  INTEGER CLASS TOTAL 7
  DESCRIPT
    "200 samples accquired after trigger.
     Classes correspond to impact cells"

DATA

 # DT 100 112 108 130 ... (200 data) ... 97 101
   CLASS 1 #

 # DT 102 104 128 110 ... (200 data) ... 127 88
   CLASS 6 #
   ...
   ... (more recepts) ...
   ...
 # REM "cell 4 has been damaged"
   DT 82 87 1008 93 ... (200 data) ... 111 113
   CLASS 4 #
   ...
   ... (more recepts) ...
   ...
STRUCT

 BINARY CLASS LENGTH 3 BCD CODE

DATA

 # DT 101 144 314 99 ... (200 data) ... 117 96
   CLASS 101 #
   ...
   ... (more recepts) ...
   ...
STRUCT

 UNKNOWN CLASS

DATA

  # DT 102 104 128 110 ... (200 data) ... 127 88 #

  # DT 101 105 110 115 ... (200 data) ... 99 100 #
  ...
  ... (more recepts)
  ...
  # DT 100 100 99 98 ... (200 data) ... 187 203 #

END
```

Fig. 4. LDP example file.

As an example of this sensor data storing scheme, we will show the files system used for storing the two kinds of information: original and processed, used in our latest research work at IAI.

(1) Raw data

⟨filename⟩.J

Description of raw data structure. It is an ASCII file with, at least, the following information:

$\langle n_r \rangle$ total number of recepts stored

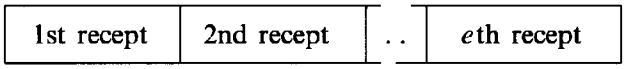
$\langle s \rangle$ number of sensors

⟨n_1⟩ data length of sensor 1
⟨n_2⟩ data length of sensor 2
. . . .
⟨n_s⟩ data length of sensor s

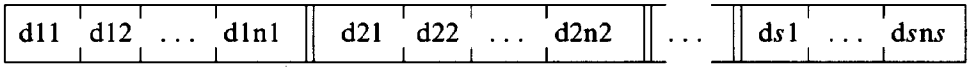
It could include identifiers of each sensor, as well as acquisition conditions (sampling rate, etc.).

⟨filename⟩.R

This file contains the raw data itself. It should be, for instance, a *file of integer* because raw sensor data always come from analog to digital converters (ADC). The organization of this file is very sample. A list of recepts:



where each recept contains the time series obtained by the all sensors:



(2) Vectors of features

⟨filename⟩.I

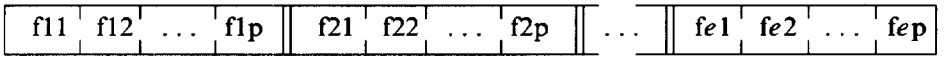
This ASCII file contains the number of processed recepts (vectors of features) stored and the length of these vectors (number of features extracted from the recept):

⟨*nr*⟩
⟨*nf*⟩

It could include information describing the process performed over the sequence of recepts, feature identifiers, etc.

⟨filename⟩.F

Contains all the vectors of features. It is a *file of real*, because a feature is the result of some arbitrary processing over raw data or previous features. Its structure is just a list of vectors of features:



Class information

⟨filename.C*⟩

The class associated with each recept, either original (files *.R and *.J) or processed (files *.I and *.F) may be stored in simple files of byte, to easily handle finite classification tasks.

We can maintain several classification files in order to admit different perception objectives from a common experimental base. For instance, the learning stage in recognition of words from digitized speech would require a file called voice.CW, different from the file voice.CS, used to train the classifier to detect the speaker from the same digitized speech sequence of recepts.

4. Concluding remarks

A remarkable fact of the proposed approach is the fusion of sensor data acquisition and processing. Given a sensor system we provide a definite features extraction program, which will be automatically executed after every new recept acquisition. Any module (e.g., pattern classifiers, statistical packets, graphic representation programs, etc.) can access data transformed according to its own needs, e.g., significative features extraction, redefinition of

recept classes, normalization or discretization of variables, and so on. Sensors virtually generate significative features instead of raw data.

Unfortunately, the main advantage of this approach (coherent data access and processing for state identification) is unrealistic for real-time applications using general purpose computers. An interesting solution is to delegate acquisition and hard processing stages to *digital signal processors* (DSP). The master computer instructs the DSP (sending it a certain command string) for collecting raw data and executing the appropriate processing on them, generating recepts in the form of vectors of features in a shared memory area. These (optionally) processed recepts may be used both in design stages (feature selection, evaluation of classification tools performance, etc.) and for real-time operation identifying the class or state of unknown recepts.

It is reasonable to take advantage of object oriented extensions (e.g., streams) of most popular languages to implement sensor data structures as well as to define the storing format and drivers both for real sensors and data files.

Finally, to approach a new sensor integration task, it is essential to know a suitable estimation of certain design parameters required for development stages and decisive for system performance. Some of them are:

(1) Which concepts, states or classes are to be recognized from sensor information?

(2) Which sensors (number and nature) seem to be appropriate for this task? What is their proper structure?

(3) Are there any bibliographic references about useful features to extract from raw sensor data?

(4) Is it possible to build a training set of recepts covering statistically all possible situations for the system?

(5) Is it necessarily a real-time work of the sensor system? In that case, what is the performance of acquisition and processing hardware?

Acknowledgements

The authors would like to acknowledge the support provided by the Comunidad Autónoma de Madrid (CAM) under the project C198/90 and by the Comisión Interministerial de Ciencia y Tecnología (CICYT) under the project ROB89-1130-CE.

References

1 Y. F. Zheng, Integration of multiple sensors into a robotic system and its performance evaluation, *IEEE Trans. Robotics Automation*, *5* (5) (Oct.) (1989).

2 A. M. Agogino and S. Srinivas, Multiple sensor expert systems for diagnostic, reasoning, monitoring and control of mechanical systems, *Mech. Syst. Signal Process.*, *2* (2) (1988) 165–185.

3 D. Guinea, A. Ruiz and L. J. Barrios, Multi-sensor integration: an automatic feature extraction and state identification methodology for tool wear estimation, *Proc. First CIRP Workshop of Intelligent Manufacturing Systems, Budapest, Hungary, Mar. 6–8, 1991*, Seminars on Learning in IMS, pp. 159–175.

4 A. Ruiz, Mecanismos de integración multisensorial: un sistema de percepción arficial (Multi-sensor information schemes: an artificial perception system), *Doctoral Thesis*, Universidad Complutense de Madrid, Spain, Nov. 1990.

5 M. W. Sobolewski, Percept knowledge-base systems, in I. Plander (ed.), *Artificial Intelligence and Information-Control Systems of Robots*, Elsevier, Amsterdam, 1987.