

ELSA: An Intelligent Multisensor Integration Architecture for Industrial Grading Tasks

by

Michael David Naish

B.E.Sc., University of Western Ontario, 1996

B.Sc., University of Western Ontario, 1996

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF APPLIED SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

(Department of Mechanical Engineering)

We accept this thesis as conforming
to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

November 1998

© Michael David Naish, 1998

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of MECHANICAL ENGINEERING

The University of British Columbia
Vancouver, Canada

Date DECEMBER 9, 1998

Abstract

The Extended Logical Sensor Architecture (ELSA) for multisensor integration has been developed for industrial applications, particularly, the on-line grading and classification of non-uniform food products. It addresses a number of issues specific to industrial inspection. The system must be modular and scalable to accommodate new processes and changing customer demands. It must be easy to understand so that non-expert users can construct, modify, and maintain the system.

The object model used by ELSA is particularly suited to the representation of non-uniform products, which do not conform to an easily specified template. Objects are represented by a connected graph structure; object nodes represent salient features of the object. Object classifications are defined by linking to primary features, each primary feature may be composed of a number of lower-level subfeatures.

Sensors and processing algorithms are encapsulated by a logical sensor model, providing robustness and flexibility. This is achieved by separating sensors from their functional use within a system. The hierarchical structure of the architecture allows for modification with minimal disturbance to other components.

The construction methodology enables domain experts, who often lack signal processing knowledge, to design and understand a sensor system for their particular application. This is achieved through a formal design process that addresses functional requirements in a systematic way. Each stage involves the extraction and utilization of the user's expert knowledge about the process and desired outcomes. Specification of the requirements leads to the identification of primary features and object classifications. Primary features are expanded into subfeatures. Logical sensors are then chosen to provide each of the features defined by the object model; this in turn determines what physical sensors are required by the system. The object classifications determine the rulebase used by the inference engine to infer process decisions.

Contents

Abstract	ii
Table of Contents	iii
List of Tables	ix
List of Figures	x
Acknowledgements	xii
1 Introduction	1
1.1 Traditional Industrial Inspection and Grading	1
1.2 Automated Inspection	2
1.3 The Need for Multiple Sensors	3
1.4 The Industrial Problem	3
1.5 Potential Benefits of a New Approach	4
1.6 Project Scope and Objectives	5
1.7 Thesis Outline	6
2 Literature Review	7
2.1 Multisensor Integration vs. Multisensor Fusion	7
2.2 Advantages of Multisensor Integration	8
2.3 Multisensor Integration Architectures	8
2.3.1 Logical Sensor-Based Architectures	11

2.4	Sensor Technologies	14
2.5	Sensor Standards	15
2.6	Industrial Applications	17
2.7	Uncertainty and Accuracy	20
2.8	Object Modelling	22
3	Object Modelling	27
3.1	Introduction	27
3.2	Rationale	27
3.3	Approach to Modelling	28
3.4	Model Structure	29
3.4.1	Classification Layer	31
3.4.2	Feature Layer	31
3.5	Properties of Objects	32
3.5.1	Physical Properties	32
3.5.2	Relational Properties	32
3.6	Model Components	33
3.6.1	Object Nodes	33
3.6.2	Unconditional Links	35
3.6.3	Fuzzy Links	35
3.6.3.1	Linguistic variables	35
3.6.3.2	Membership functions	37
3.7	Model Definition	38
3.8	Summary	39
4	System Architecture	40
4.1	Introduction	40
4.2	Logical Sensors	41
4.2.1	Logical Sensor Characteristics	45
4.2.2	I/O	46

4.2.2.1	I/O Controller	46
4.2.2.2	Data Input	47
4.2.2.3	Data Output	47
4.2.2.4	Control Input	48
4.2.2.5	Control Output	48
4.2.3	Controller	48
4.2.3.1	Logical Sensor Controller	49
4.2.3.2	Local Exception Handling	50
4.2.3.3	Local Knowledge Base	51
4.2.4	Programs	51
4.2.4.1	Device Drivers	51
4.2.4.2	Processing Algorithms	52
4.3	Integration	52
4.3.1	Integration Controller	53
4.3.2	Validation	53
4.3.3	Diagnostics	54
4.3.4	Exception Handling	54
4.3.4.1	Error Classification	55
4.3.4.2	Error Recovery	56
4.4	Inference Engine	56
4.4.1	Rule/Knowledge Base	58
4.5	Post Processing	59
4.6	Summary	59
5	Construction Methodology	60
5.1	Problem Definition/Requirements Specification	60
5.2	Object Model Development	62
5.3	Logical/Physical Sensor Selection	64
5.4	Rulebase Definition	67
5.5	System Implementation	70

5.6	Modification and Refinement	72
5.7	Summary	74
6	Application Examples	75
6.1	Can Defect Detection	75
6.1.1	Background	75
6.1.2	Problem Definition/Requirements Specification	76
6.1.3	Object Model Development	78
6.1.4	Logical/Physical Sensor Selection	79
6.1.5	Rulebase Definition	81
6.1.6	Summary	84
6.2	Herring Roe Grading	84
6.2.1	Background	84
6.2.2	Problem Definition/Requirements Specification	86
6.2.3	Object Model Development	88
6.2.4	Logical/Physical Sensor Selection	89
6.2.5	Rulebase Definition	97
6.2.6	Summary	98
6.3	Discussion	99
7	Concluding Remarks	101
7.1	Summary and Conclusions	102
7.2	Recommendations	103
	References	106
	Appendices	112
A	Object Model Class	112
A.1	Introduction	112
A.2	Class Summary	112
A.3	The Classes	113

CNode	113
CObjectNode	115
CClassificationNode	119
CObjectProperties	121
CPhysicalProperties	122
CRelationalProperties	125
B Extended Logical Sensor Class	128
B.1 Introduction	128
B.2 The Class	128
CELSBase	128
C Fuzzy Variable Class	136
C.1 Introduction	136
C.2 Class Summary	136
C.3 The Classes	138
CFuzzyDegree	138
CFuzzyVariable	140
CFuzzyMember	145
CFuzzyGeomMember	147
CFuzzyTriMember	150
CFuzzyTrapMember	153
CFuzzyArrayMember	156
D Rulebase Classes	159
D.1 Introduction	159
D.2 Class Summary	159
D.3 The Classes	160
CFuzzyClause	160
CFuzzyRule	162
CFuzzyRuleBase	166

E Support Classes	170
E.1 Introduction	170
E.2 Class Summary	170
E.3 The Classes	171
Max	171
Min	171
CElement	172

List of Tables

2.1	Examples of sensors for industrial inspection classified by type.	14
3.1	Components of object node for feature representation.	33
4.1	Summary of Extended Logical Sensor components.	44
4.2	Standard logical sensor control commands.	49
4.3	Standard logical sensor error conditions.	50
6.1	Summary of herring roe grades.	85
6.2	Dependencies of roe classifications on primary features.	91
6.3	Calculation of structured light geometry.	95
A.1	Enumeration of object types.	116
A.2	Enumeration of property types.	124
B.1	Abstract and derived types.	131
B.2	Enumeration of ELS control commands: CommandID.	132

List of Figures

2.1	Basic components of a logical sensor.	12
2.2	Instrumented Logical Sensor.	13
2.3	Networked smart transducer model.	16
2.4	Distinction between bias error and precision error.	21
2.5	Presumed processing stages in human object recognition.	24
2.6	Four steps in object grading.	24
2.7	Model used to recognize cranial CT images.	25
3.1	Graph structure for object representation.	30
3.2	Object node for feature representation.	34
3.3	Classification node.	34
3.4	Effect of hedge <i>definitely</i>	37
3.5	Membership function used to represent confidence in the detection of a particular feature.	38
4.1	Overview of Extended Logical Sensor Architecture.	42
4.2	Basic components of an Extended Logical Sensor.	43
4.3	The Inference Engine used by ELSA.	57
5.1	Overview of construction methodology.	61
5.2	Object model development methodology.	63
5.3	Methodology for the development of the ELS hierarchy.	66

5.4	Methodology for the definition of the rulebase for object classification using the object model.	69
5.5	Methodology for the definition of the decision rulebase based on object classifications.	70
5.6	Membership function used to represent confidence that an object is of a particular classification.	71
6.1	Examples of canner's double seam defects — side view.	77
6.2	Examples of canner's double seam defects — top view.	77
6.3	Object model for metal can inspection.	79
6.4	Logical sensor hierarchy for metal can inspection.	80
6.5	Full view of can sides reconstructed from four viewpoints.	81
6.6	Rules used to identify the classification of metal cans from primary features.	82
6.7	Rules used to decide whether to reject cans based on object classifications.	82
6.8	Membership functions used for classification of metal can defects.	83
6.9	Prototype herring roe grading system.	86
6.10	Object model for herring roe grading.	90
6.11	Logical sensor hierarchy for herring roe grading.	92
6.12	Examples of herring roe classification grades imaged on-line under diffuse light conditions.	93
6.13	Geometry of structured light used for acquisition of 3D features.	94
6.14	Basic geometry for reconstruction of 3D profile information using structured light. .	95
6.15	Example of herring roe classification grades imaged on-line under structured light conditions.	96
6.16	Rules used to identify herring roe grades from primary features.	97
6.17	Rules used to determine decisions about how roe should be handled based on object classifications.	98
6.18	Membership functions used for classification of herring roe grades.	99
C.1	Triangular membership function.	151
C.2	Trapezoidal membership function.	154

Acknowledgements

This thesis is a reality due to the help, guidance, and support of many people. Foremost, I would like to extend thanks to Dr. Elizabeth Croft for providing me with an opportunity to study at UBC. Her door always open, red pen ever ready, I thank her for supervision and guidance. Together with Dr. Clarence de Silva, I also thank her for providing me with an excellent working environment in the Industrial Automation Laboratory.

I am grateful to Dr. Beno Benhabib of the University of Toronto for providing insightful, helpful, and always interesting comments on my work during his sabbatical on the West Coast. Offering a critical outside perspective, he contributed greatly to the improvement and validation of this work.

The financial support provided by the Natural Sciences and Engineering Research Council of Canada and the Gordon M. MacNabb Scholarship Foundation is gratefully acknowledged. Additional support was provided by the Garfield Weston Foundation.

Thanks to my friends and colleagues in the Industrial Automation Laboratory and the Neuro-motor Control Laboratory who, despite extending my stay in Vancouver, made my time here much more rewarding and enjoyable.

I thank my family: My parents, David and Sharon Naish, who have been unfailing in their love, support, encouragement, and assistance throughout my life. I am eternally indebted to them. Also, my sisters Jennifer and Victoria who provide me with a refreshingly unscientific view of the world. I am a better person for them.

Finally, I must thank Ana Luisa Trejos Murillo whose love, understanding, and encouragement has helped immensely through the final months of this thesis. Thank you for your patience.

Chapter 1

Introduction

1.1 Traditional Industrial Inspection and Grading

The ability to consistently produce high-quality products is important to the success of manufacturing and processing operations. Traditional quality assurance methods have often relied on human operators who use visual cues in order to determine product quality. Such methods are tedious, time-consuming, and inconsistent.

For example, for many food products, grading is performed by seasonal workers. The shifts are often long, the working conditions difficult, and there are often time constraints imposed to ensure product freshness. Grading is often a dull, repetitive task that requires long periods of concentration. Performance, and hence product quality, often degrades over the period of a shift. Furthermore, value-conscious consumers are demanding an increasing number of product classifications of high-consistency. Unfortunately, grading consistency is inversely proportional to the number of grades — as the number of grades increase, consistency decreases.

Most human multi-factor grading decisions are based on the subjective interpretation of visual information and cues from other senses (e.g. smell, firmness, weight). Thus, the characterization of grading classifications is often difficult and the ability to make repeatable decisions is hampered. This problem is compounded by the nature of natural and biologically formed products which generally do not have crisp, ideal templates [1], but rather, exhibit non-standard and non-uniform characteristics. For example, products such as fish, apples, potatoes, chicken, tomatoes, and other

types of produce may, even within a single classification or grade, vary widely in appearance. The problem is further compounded by variations in the product characteristics within a species, region, or industry. Recent trends are reducing the tolerances for acceptable products while the number of varieties and overall demand for products continue to increase. Industry has reacted by turning to automation to address these grading and quality assurance needs.

1.2 Automated Inspection

The majority of inspection and grading tasks require the acquisition and processing of visual information. In the context of industrial automation, this is handled using a machine vision system. If required, visual information may also be augmented with data from other sensors to properly assess product quality.

Machine vision systems offer a number of potential benefits to industries which rely on manual quality assurance. Since most production facilities run continuously, defects may go undetected if an inspector looks away or experiences a lapse in concentration. On the other hand, a machine vision system can guarantee that 100% of the objects leaving the system are inspected. The rate of defect detection may fall below 100% but, by increasing the inspection rate, it may achieve higher defect detection rates than a human inspector.

In addition to the reliability and repeatability of the grading system, industrial users require the ability to modify the grading scheme to meet changing market demands and customer criteria. An automated system has a consistent internal representation of product and quality classifications. This representation may be redefined by adding or removing information which governs the decision making process. Such 'global' changes offer increased consistency and flexibility over trained workers who each maintain slightly different interpretations of quality [2].

An automated system has the ability to collect on-line data about the process. This data may be used to close the control loop of the system. For example, the process parameters could be adjusted in response to fluctuations in the defect rate. This information may also be useful to marketing and sales departments who could tie the value of the product to documented quality levels.

Finally, by reducing or eliminating the need for manual inspection, labour costs would be reduced. Automation may reduce burden of maintaining a trained workforce for seasonal industries

— either in retaining skilled workers or providing training for new-hires at the start of the season. An automated system may also provide the potential for increased production speeds.

The choice to adopt an automated solution must be balanced against the inherent disadvantages. Industrial systems require a capital expenditure for initial acquisition and installation that can be significant. For some tasks, manual labour or a combination of mixed automation and labour may be more cost effective. Humans are easily trained and can adapt to new conditions and criteria quickly. This contrasts with the setup and ‘training’ of an automated system which may be time-consuming, complex, and difficult to adapt. Also, while automation typically surpasses human capabilities for product throughput, humans are better able to handle unexpected events and tasks which involve a combination of inspection and handling operations, such as the patching of salmon cans [3].

1.3 The Need for Multiple Sensors

Systems which have attempted to make multi-factored grading decisions on the basis of information from a single sensor have met with limited success. Despite the richness of information available from a colour camera, such a device can only produce a two-dimensional array of intensities from a single viewpoint. Features that may have a significant influence on the assigned grade may be occluded from view or require depth information for detection. Often, it is desirable to combine visual information with data from other sensors to improve the outcome. Possibilities include the combination of vision with simpler sensors, such as load cells and thermocouples, or the use of multiple cameras to eliminate occlusion or produce depth maps (through stereo vision algorithms). In addition to the advantages of using complementary information as mentioned above, multiple sensors may also provide redundant information to improve the accuracy and robustness of a system.

1.4 The Industrial Problem

The role of machine vision and multisensor integration is becoming widely accepted in the food processing industry [4, 5]. Intelligent multisensor systems are intended to provide complementary qualities to the industrial user; namely, the repeatability and reliability of automation together with the feature discrimination, classification capability, and adaptability of humans. However, there are

two main problems with current industrial multisensor systems.

First, in an effort to use multiple sensors to improve process performance, many systems have been constructed in an ad-hoc fashion. Pieces are added as new technology is acquired, often with the need to redesign significant portions of the existing system to facilitate the integration of the new sensors. Such systems lack a formal architecture and are typically designed by experts in machine vision and/or systems integration. This is a general problem for systems that have been designed and constructed for a specific task or operation. For industries competing in dynamic markets that require systems which can adapt to changing needs for speed, feature recognition, accuracy, and product differentiation, this approach is problematic.

Second, much of the success of machine vision and multisensor systems is dependent upon the ease of use of these systems for industrial users. Such users may understand the process but not the details of the sensor technology. In order to achieve full acceptance, the associated sensor and artificial intelligence technologies must become transparent to the end user, so that process experts in the food industry do not have to understand the technical details. A completely transparent system is likely many years away. As a result, opportunities exist to develop systems which work towards this goal, while achieving the proper balance between utility and ease of use. Such systems should be orderly, comprehensible, and simple.

1.5 Potential Benefits of a New Approach

There is currently no accepted formal approach for the design and construction of a multisensor integration system for industrial inspection. An open and scalable architecture will enable industrial users to design systems which inherently reduce the risk of obsolescence. Systems may be reconfigured, modified, and adapted to respond to changing requirements and advances in sensor technology.

By organizing the system in a manner that industrial users can understand, these same people can specify, configure, and maintain their own systems, without the need to retain outside experts. This places the power to define and modify the process with those that understand it best, reducing the need to transfer process knowledge to automation experts.

1.6 Project Scope and Objectives

This work represents the first stage of an initiative to develop both a methodology to construct multisensor integration systems in a systematic way and to provide the tools required to do so. This thesis focuses on the development of the Extended Logical Sensor Architecture (ELSA) to allow for the systematic construction of a multisensor integration system for industrial tasks. In doing so, the underlying structure, the major components dealing with sensing and inference issues, the object representation, and the construction methodology are presented. It should be noted that while inspection is the focus of this work, it is intended to be applicable to a variety of automation tasks which may benefit from a multiple sensor perception system. Other potential applications include material handling, assembly, and machining operations.

To address the industrial needs outlined in the sections above, ELSA is presented as new, open architecture approach for intelligent multisensor integration in an industrial environment. The specific objectives of this thesis are detailed below:

1. To provide a modular and scalable architecture which serves as a robust platform for intelligent industrial sensing applications.
2. To specify an encapsulation of physical devices and processing algorithms.
3. To specify a data representation scheme which allows for the quantification of deviations from an ideal model.
4. To ensure that the data representation scheme provides the user with insight as to how the system is structured and how the sensor information is used to make decisions.
5. To provide a robust exception handling mechanism to ensure the reliability of an implementation of this architecture.
6. To ensure that the architecture is applicable to a broad range of industrial applications, especially those involving non-uniform product grading.

1.7 Thesis Outline

The structure of this thesis is summarized in the outline below:

Chapter 1 Introduction: This introductory chapter.

Chapter 2 Literature Review: Introduces literature from a variety of topics related to the problem of industrial multisensor integration. Topics include: visual inspection, multisensor integration, integration architectures, and object representation.

Chapter 3 Object Modelling: Describes how objects are represented using an object model which also provides a basis for sensor selection and inference engine implementation.

Chapter 4 System Architecture: Presents the overall structure of the Extended Logical Sensor Architecture (ELSA). Each of the major components, particularly those for sensing and inference, are detailed.

Chapter 5 Construction Methodology: Discussion of the ELSA approach to the design of, and selection of components for, a multisensor integration system for industrial tasks.

Chapter 6 Application Examples: Presents two illustrative example applications taken from industry. These serve to demonstrate how a system is implemented within the ELSA architecture.

Chapter 7 Conclusions and Recommendations: Concludes the thesis with a summary which highlights the contributions of this work. Suggestions for future improvements to ELSA and related systems are also given.

Chapter 2

Literature Review

Herein, the relevant literature dealing with multisensor integration, visual inspection, and object modelling is presented. The majority of the work in the area of multisensor integration has focused on issues specific to mobile robotics and target tracking applications; however, many aspects of these systems are applicable to visual inspection. The object modelling literature, on the other hand, is closely tied to the image processing and machine vision research. This work is therefore a synthesis of the ideas from these different fields, applied to the problem of sensor integration for industrial inspection.

2.1 Multisensor Integration vs. Multisensor Fusion

Multisensor integration and multisensor fusion are closely related. The role of each in the operation of intelligent machines and systems is best understood with reference to the type of information that the integrated multiple sensors can uniquely provide to the system. Multisensor integration refers to the synergistic use of the information provided by multiple sensory devices to assist in the accomplishment of a task by a system [6]. The somewhat more restricted notion of multisensor fusion refers to any stage in the integration process where there is an actual combination (or fusion) of different sources of sensory information into one representational format. Fusion may occur at a low-level (signal), mid-level (pixel), or high-level (feature or symbol) of representation. These definitions serve to distinguish the system-level issues relating to *integration* of multiple sensory devices at the architecture and control level from the more specific mathematical and statistical issues presented

by the actual *fusion* of sensory information. In this work, the focus is on integration.

2.2 Advantages of Multisensor Integration

Automated systems which attempt to make multi-factored decisions about non-uniform products on the basis of information from a single sensor have had limited success. Often, there is simply inadequate data for a proper product assessment. The transition to multiple sensors can extend the capabilities and improve the robustness of existing systems.

A system which employs multiple sensors may enjoy several advantages over single sensor systems [6]. The primary advantages are: information can be obtained more accurately and features undetectable with individual sensors may be perceived in less time and with less cost. Of these, redundant and complementary information are the most certain.

Redundant information is acquired by a group of sensors (or a single sensor over time); each sensor perceiving the same features in the environment. By integrating and/or fusing this information, the accuracy of the system can be increased by reducing the overall uncertainty. Redundant sensors also serve to increase the robustness of the system in the event of sensor failure. Complementary sensor groups, on the other hand, perceive features in the environment that are imperceptible to individual sensors. Each sensor provides a subset of the required feature space; these feature subsets are combined to obtain the intact feature.

Little published work has been done in the area of non-specialized, sensor integration architectures for industrial applications. Section 2.3 provides a brief review of general sensor integration architectures.

2.3 Multisensor Integration Architectures

A system architecture provides a framework upon which individualized systems can be built and adapted. For complex systems, an architecture is essential to ensure that the system is comprehensible, robust, and that it is easily extensible. An architecture for sensor integration systems must provide the following components:

- Data structure and communication protocols.
- Resolution of information from sensors.
- Data fusion/integration engine.
- Exception handling.
- Decision making (inference from sensory information).
- Control mechanism or method of utilizing system output.

A number of different architectures have been developed for the purpose of multisensor integration, each for a specific application such as mobile robot navigation and control, autonomous guided vehicles, military target tracking, and industrial control systems. While, on the whole, these architectures are not directly applicable to the task of industrial inspection and classification, each of the following examples presents some aspects which are potentially useful to this problem.

Architectures developed for mobile robot navigation and control are primarily concerned with prioritizing objectives and ensuring that high priority (real-time) objectives are met. Brooks' subsumption architecture [7] utilizes a number of different layers to ensure that while performing a high-level task, low-level operations continue to perform. For example, the task of chasing a mouse around a room is overridden by the goal of avoiding obstacles and this, in turn, is overridden by the need to maintain balance. Sensory information may be used differently by each layer; only those sensors which perception processing identifies as extremely reliable are used to maintain a central representation.

An action-oriented perception paradigm is utilized by the SFX architecture developed by Murphy [8, 9]. Robot actions are decomposed into a motor behaviour and the perception of some object or event which drives the behaviour. Perception considers both the percept being sensed and a measure of certainty in the sensing. A cognitive science model proposed by Bower [10] is used as the basis for discordance-based sensor fusion to combine information from multiple sensors. There are four fusion modes as follows:

1. Complete sensor unity (*fusion* of sensor data). In this mode, sensor data is fused without a mechanism for detecting discordances. Sensory information is tightly coupled such that discordances do not arise.
2. Awareness of discordance where recalibration is possible (*integration* of sensor data). Here, the discordance between sensors is reconciled by recalibration of the offending sensors.
3. Awareness of discordance where recalibration is not possible (*comparison* of sensor data). In this case, sensors providing erroneous data are temporarily suppressed.
4. No unity at all (*independent* use of sensor data). Sensors observe attributes without any spatial correspondence. Here, sensor data is used independently.

In Murphy's architecture, sensing failures are handled by error classification and error recovery modules. Classification of the error type and source is attempted using a modified generate-and-test procedure. Once the error source is identified, the error recovery module selects a predefined recovery scheme to either repair or replace the current sensing configuration.

Lee [11,12] has developed the Perception Action Network (PAN) architecture which provides a mechanism for dealing with uncertainty in the process of data fusion. Feature Transformation Modules (FTM), Data Fusion Modules (DFM), and Constraint Satisfaction Modules (CSM) are placed along the connections between logical sensors (to be defined shortly). These modules define relationships which allow the perception net to reduce uncertainties through data fusion and constraint satisfaction, in addition to identifying possible biases.

Architectures developed for Autonomous Guided Vehicles (AGVs), are concerned with issues similar to those of mobile robotics. The approach taken by Draper *et al.* [13] in the development of the sensor integration system for the Mobile Perception Laboratory (MPL) was to focus on the types of information required to perform a task and the representations needed to express them. This shifts the use of data from multiple sensors from low-level fusion to higher-level integration. Another approach to an architecture for AGVs is presented by Shafer, Stentz, and Thorpe [14].

Sensor integration research within the military has focused on target tracking applications. The major issues here are proper synchronization, communication, and routing between sensor systems that are widely distributed. Architectures which have been developed include those by Iyengar *et*

al. [15,16] based on a multilevel binary de Bruijn network (MBD), Klein [17], and the object-oriented approach taken by Queeney and Woods [18].

The industrial operating environment is often quite different from the operating environments of mobile robots, AGVs, and target tracking systems. For these, the environment is assumed to be unstructured and largely unknown. A priori knowledge of lighting conditions, expected objects, obstacles, and failure modes is often unknown or unavailable. In an industrial context, many of these factors may be controlled. Additionally, industrial systems do not have to address the behaviour issues of mobile robotics.

There are few integration architectures that have been developed to address problems specific to the industrial environment. One exception is the HINT architecture developed by Alarcón *et al.* This is a generic architecture for plant-wide industrial control [19]. It aims to support the integration of different artificial intelligence techniques to provide solutions to process control problems that currently require human expertise. While not directly applicable to industrial inspection and grading tasks, it presents some interesting ideas. In particular, the proposed methodological approach and hierarchical structure of the framework are useful starting points for the design of a new architecture.

2.3.1 Logical Sensor-Based Architectures

Sensors are one of the principal building blocks of a multisensor integration architecture. The data provided by sensors may be used as input to processing algorithms which combine and convert the data into higher level representations of the information. One approach that is well suited to the incorporation of sensors into a multisensor integration architecture is the logical sensor model.

A logical sensor (LS) is an abstract definition for a sensor. Logical sensors were first defined by Henderson and Shilcrat [20] and later broadened to include a control mechanism by Henderson, Hanson, and Bhanu [21], Figure 2.1. This definition provides a uniform framework for multisensor integration by separating physical sensors from their functional use within a system. Logical sensors are used to encapsulate both physical sensors and processing algorithms. This encapsulation defines a common interface for all sensor types allowing the straightforward addition, removal, and replacement of sensors within the architecture.

Using this definition, physical sensors such as load cells, thermocouples, cameras, and laser

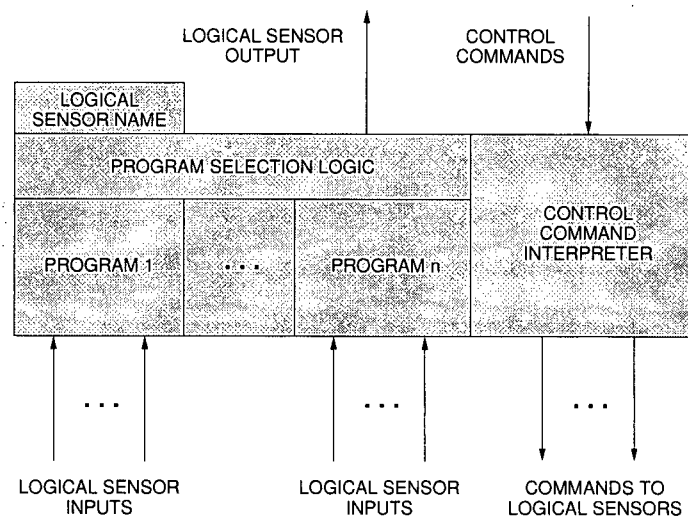


Figure 2.1: Basic components of a logical sensor as proposed by Henderson *et al.* [21].

range-finders may be represented. The data from these sensors may also be combined and processed using a variety of available algorithms. In this way, logical sensors, such as a line detector, which do not physically exist, may be made available to the user. Output from a variety of logical sensors may be combined to extract complex features. Physical sensors may be replaced or added without disturbing the entire system — only the associated logical sensor need change.

The logical sensor model provides a control structure which allows for the selection of a different program (which may rely on different sensor inputs) should the sensor performance prove unacceptable. Control commands are generated from higher-level logical sensors.

The original Logical Sensor Specification (LSS) handles error conditions in a simple manner. An 'acceptance test' is used to judge each input. Inputs which pass the test are accepted and used; those that fail are rejected. Rejection results in the system attempting to obtain input from one of a number of alternate inputs. When all alternatives are exhausted, the sensor fails. The source of the error is not determined. Replacement rather than recovery is the only method of error handling.

Weller, Groen, and Hertzberger adopted the logical sensor concept and developed an architecture which uses a hierarchy of sensor modules [22]. Before replacing sensor modules in an effort to obtain correct input, an attempt is made to locate the cause of the error. If found, recovery is attempted through the adjustment of sensor parameters and/or input to the sensor. This approach

requires that the sensor itself contain expert knowledge for both the detection and isolation of error conditions, and the rules upon which the recovery strategy is based. This concept was further refined by Groen, Antonissen, and Weller when applied to a model-based robot vision system [23].

Dekhil and Henderson extended the concepts introduced by Weller *et al.* and introduced Instrumented Logical Sensor Systems (ILSS) [24–27]. The application was again mobile robot navigation. The ILSS, as shown in Figure 2.2, is an extension of the LSS. The primary difference between ILSS and LSS is the addition of components which provide mechanisms for on-line monitoring and debugging. These mechanisms are designed to increase the robustness of the sensor. For example, monitors use a set of rules to validate the output of the ILSS. The user is alerted to any undesired results. Local embedded testing is used for on-line checking and debugging purposes. These tests operate by generating input data designed to check different aspects of the module. The data may also be directed at other sensors to enable the testing of a group of modules. A set of rules and constraints is used to examine the resulting sensor output.

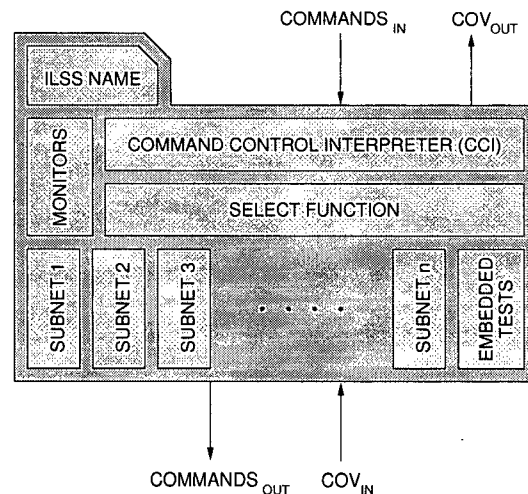


Figure 2.2: Instrumented Logical Sensor [27].

Using the ILSS, data from physical sensors may be combined and processed using a variety of algorithms to create sensors which do not physically exist. A sensor system may be constructed which can extract complex high-level features. These features form the basis of the object representation for recognition and classification.

2.4 Sensor Technologies

Sensors are most often classified in accordance with the type of physical phenomenon that is detected and the subject of measurement. Often sensors are developed to satisfy particular objectives. A large number of sensors exist for an ever increasing number of applications which utilize sensor technologies [28]. Table 2.1 presents examples of sensors which are particularly suited to industrial inspection tasks.

Table 2.1: Examples of sensors for industrial inspection classified by type.

Sensor Type	Detection Data	Detector
Visual	Position Distance Form Features	Cameras, position sensors, range finders, line image sensors, area image sensors.
Proximity	Proximity Spacing Inclination	Photoelectric switches, LED lasers, phototransistors, photodiodes, ultrasound sensors.
Tactile	Contact Force Pressure	Limit switch, metal detectors, strain gauges, conductive rubber, pressure sensors.
Aural	Sound	Ultrasound sensors.
Chemical	Gas Odour pH	Emission spectroscopy, chromatographs, densitometers, gravimeters, X-rays.
Dynamic	Weight Speed	Load cells, accelerometers, anemometers, LIDAR.

Typically, the sensor output signal is in the form of an electrical signal such as a voltage. The output could be in either analog or digital form. Since most current applications process sensor information using a computer, sensors which provide digital output are preferable. Analog signals are converted into a digital representation using an analog-to-digital (A/D) converter.

Device drivers are required to interface between the hardware of the sensor and the processing system. These programs serve to package data and commands in a format that may be understood by both sides. Unfortunately, there is little standardization of sensor hardware — even for devices

that perform the same task. As a result, each sensor typically requires a unique driver. This reduces the interoperability of sensor technologies.

2.5 Sensor Standards

There is currently a major effort to develop a standard for the integration of sensor technologies. This effort, led by the the National Institute of Standards and Technology (NIST) and the Institute of Electrical and Electronics Engineers (IEEE), is working toward the development of the IEEE-P1451 Standard for a Smart Transducer Interface for Sensors and Actuators. The goals of this standard are to:

- Enable plug and play at the transducer level by providing a common communication interface for transducers.
- Enable and simplify the creation of networked smart transducers.
- Facilitate the support of multiple networks.

The standard consists of four subsections: P1451.1 — Network Capable Application Processor (NCAP) Information Model, P1451.2 — Transducer to Microprocessor Communication Protocols and Transducer Electronic Data Sheet (TEDS) Formats, P1451.3 — Digital Communication and Transducer Electronic Data Sheet (TEDS) Formats for Distributed Multidrop Systems, and P1451.4 — Mixed-mode Communication Protocols and Transducer Electronic Data Sheet (TEDS) Formats. Currently, draft versions exist for P1451.1 [29] and P1451.2 [30].

P1451.1 specifies networking support for a transducer. The objective is to develop standardized connection methods for smart transducers to control networks. Little or no changes should be required to use different methods of A/D conversion, different microprocessors, or different network protocols. This objective is achieved through the use of a common object model, together with interface specifications to these components. There are two key views of the networked smart transducer, as shown in Figure 2.3.

The Smart Transducer Object Model provides two key interfaces. The first, to the Transducer Block, encapsulates the details of the transducer hardware implementation within a simple pro-

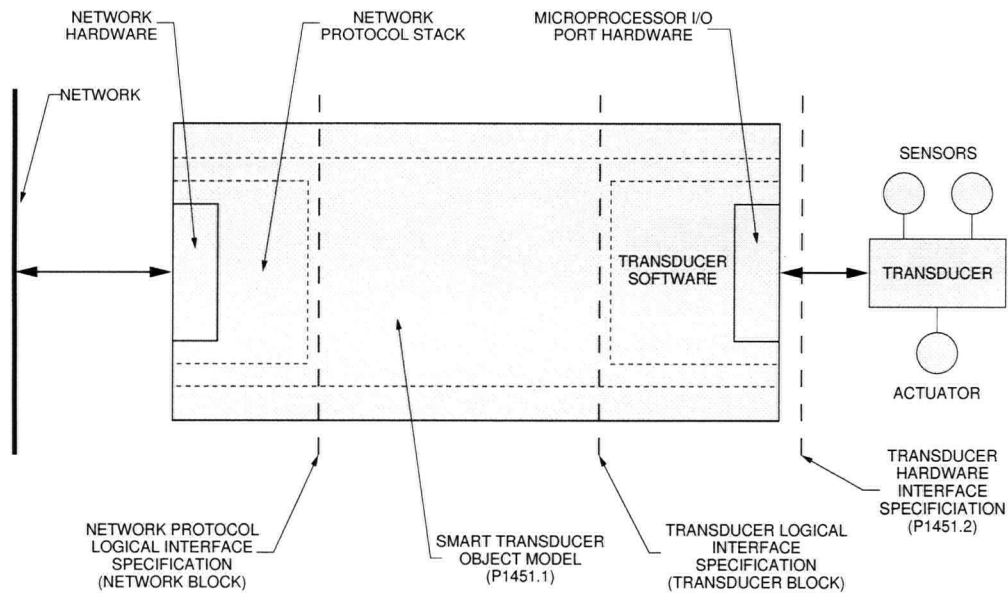


Figure 2.3: Networked smart transducer model [29]. Physical components are shown by solid lines; dotted lines indicate logical components.

programming model. This results in the sensor or actuator hardware appearing like an I/O-driver. The second is the interface to the Network Block. This interface encapsulates the details of the different network protocol implementations behind a small set of communication methods. These logical models and interfaces are used in defining a data model for the smart transducer that is supported by the network.

P1451.2 provides an interface specification to allow the interoperability of transducers. It allows for self-identification and configuration of sensors and actuators while allowing extensibility so that vendors may provide for growth and product differentiation. The Transducer Electronics Data Sheet (TEDS) provides a mechanism to specify a combination of transducer, signal conditioning, and signal conversion to the rest of the system; it does not specify the actual signal conditioning or data conversion methodologies. TEDS contains fields that fully describe the type, operation, and attributes of one or more transducers. The TEDS is physically associated with each transducer, encapsulating the measurement aspects of a Smart Transducer Interface Module.

The interfaces specified in P1451.1 and P1451.2 are optional in that an implementation may adopt one without the other. For example, if a transducer is networked but support for interoperability is not required, the P1451.1 Object Model may be used without the P1451.2 Interface

Specification. Similarly, if networking is not supported, or the network implementation is closed, P1451.1 does not have to be used to get the benefits of interoperability provided by P1451.2.

These standards have been developed to support a wide variety of transducers as well as a digital interface to access the TEDS, read sensors, and set actuators. This allows transducer manufacturers to differentiate themselves not by the supported interface(s), but by cost, feature set, and quality. Manufacturers design to a common interface which can be used by a variety of applications.

By providing a standard low-level interface, these smart transducers may be easily integrated into a sensor system. This model extends to the device level the concepts of modularity and flexibility that are desirable in a multisensor integration system. A truly encapsulated system may then be constructed — from high-level integration and processing algorithms to the low-level sensing devices.

2.6 Industrial Applications

In the area of quality assessment and assurance, machine vision is often used to gather the bulk of the required information, especially for the grading or classification of non-uniform (biological) products. Other sensors, such as scales, mechanical measurement devices, and ultrasound are employed to gather information that is used to enhance the machine vision data. Industrial systems which employ machine vision perform one or more of the following activities [4]:

- Gauging:** Performing precise dimensional measurements.
- Verification:** Ensuring that one or more desired features are present and/or undesired features are absent.
- Flaw detection:** Location and segmentation of undesired features which may be of unknown size, location, and shape.
- Identification:** Use of symbols, including alphanumeric characters, to determine the identity of an object.
- Recognition:** Use of observed features to determine the identity of an object.
- Locating:** Determination of object location and orientation.

The controlled environment of an industrial plant greatly simplifies the generic recognition problems considered by many machine vision researchers. Segmentation is simplified by knowledge about both the objects and the background against which objects must be segmented. Typical production arrangements involve the use of conveyor belts which serve to provide physical separation between the objects being transported. This separation eliminates the need for algorithms which perform well when objects are occluded; such algorithms are typically computationally expensive. In addition, structured and predictable lighting is possible, further simplifying the object recognition task by ensuring that objects appear under the same intensity of light and shadow field. This improves feature discrimination, reduces processing time, and reduces processing hardware requirements [4].

There have been a number of vision-based multisensor systems developed for quality assurance and assessment over the past decade. In most of these applications, ad-hoc methods are used to develop a sensor integration system to monitor the process. Such systems lack a formal architecture and are typically designed by experts in machine vision and/or systems integration. This can result in difficulties with the use and maintenance of the system for the everyday user. Additionally, upgrading the system to change or add additional sensors and/or requirements often requires the system to be redesigned. This is a problem for industrial users whose requirements in terms of speed, feature recognition, accuracy, and other process monitoring parameters invariably change over time. A number of examples of recent industrial systems follow.

Luzuriaga, Balaban, and Yeralan [31] have developed a system for the machine vision inspection of white shrimp. The back-lit shrimp are inspected by a single colour CCD array camera for colour and 2D shape features. Colour changes and melanosis development of the stored shrimp are used as a basis of evaluation. Additionally, the weight of each shrimp is estimated from the 2D view area. While designed for and tested in an industrial environment, this system relies on manual placement and turning of each shrimp.

A similar system for catfish feature identification was developed by Jia, Evans, and Ghate [32], though it was concerned primarily with the detection of the head, tail, and fins. These features were then used to determine appropriate cutting lines for processing.

Daley *et al.* [33] are working towards the automation of poultry grading and inspection. This system uses a colour CCD camera to obtain information regarding the HSI colour, size, and shape

of each bird. Global defects are identified with a 96% success rate; local defects are identified only about 60% of the time. This is due to the increased difficulty in extracting the local features. To properly address the problem, additional sensors are required to allow for the measurement of the surface texture and structure.

A low-cost system for fruit and vegetable grading was developed by Calpe *et al.* [34] as an alternative to expensive commercial systems. This open platform may handle up to 12 lanes simultaneously at a speed of 10 items per second. Classification is based on RGB colour information. Conveyor rollers mechanically separate and rotate the fruit. The captured image contains two lanes with four pieces of fruit in each lane. A colour index is computed for all eight pieces as the conveyor moves forward and another image is taken. In this way, four images of each piece of fruit are acquired; the rollers ensuring that the majority of surface area is considered. Currently, a grading decision is made by averaging the colour information.

Recent work in the Industrial Automation Laboratory at the University of British Columbia has involved the grading of herring roe skeins [35–38]. Images obtained with a CCD camera are processed to extract colour, contour, and curvature information. Skein weight is estimated from the 2D area using a multiple-regression estimator. Firmness is estimated from the brightness of ultrasonic echo images. All of this information is combined to determine a classification for each roe. Grading accuracy ranges from 72%–95%. Classification accuracy between Grade 1 and Grade 2 roe is about 95%; however, the system is less successful at subclassifying the Grade 2 roe into various sub-grades. Additional sensors are required to improve the overall performance of the system.

Other applications which make use of sensory information for grading and classification include potato grading [39], shrimp inspection [31], material surface inspection [40], printed circuit board inspection [41], and visual inspection of unsealed canned salmon [42].

A number of proprietary industrial systems exist for product inspection and classification. These include the QualiVision system from Dipix Technologies Inc. for the quality control of bakery and snack food products. This system uses 3D imaging to assess product consistency to 10 microns [43]. Lumetech A/S has developed the Fisheye Waterjet Portion Cutter for trimming and portioning fish fillets [44]. Lullebelle Foods Ltd. utilizes a cell-based vision system to eject unripe blueberries from the processing line [45]. Key Technologies Inc. offers the Tegra system for grading agricultural

products according to size and colour [46]. Typically, such systems sort products based on 1–2 discrete thresholds.

2.7 Uncertainty and Accuracy

There are a number of standard terms [47] which may be used to describe the validity of sensor data and the analysis of uncertainty. As the use of this terminology has not been consistent in the literature [48], a brief review follows:

Error is defined as the difference between the measured value and the true value of the measurand, as illustrated by Equation (2.1).

$$\text{error} = \text{measured value} - \text{true value} \quad (2.1)$$

There are two general categories of error which may be present: *bias errors* (systematic or fixed errors) and *precision errors* (random errors) [49]. Both degrade the validity of the sensed data, though the causes of each are different and each is minimized in a different manner.

Bias errors are consistent, repeatable errors; however, they are often not obvious and considerable effort is usually required to minimize their effects. There are three forms of bias error. The first, *calibration error*, is the result of error in the calibration process, often due to linearization of the calibration process for devices exhibiting non-linear characteristics. The second source of bias error is *loading error*. This is due to an *intrusive* sensor which, through its operation, alters the measurand. Loading error may be avoided through the use of *nonintrusive* sensors. Lastly, a bias error may result from the sensor being affected by variables other than the measurand. Bias errors are defined by Equation (2.2).

$$\text{bias error} = \text{average of readings} - \text{true value} \quad (2.2)$$

Precision errors are caused by a lack of repeatability in the output of the sensor. These are

defined by Equation (2.3). Bias errors and precision errors are contrasted in Figure 2.4.

$$\text{precision error} = \text{reading} - \text{average of readings} \quad (2.3)$$

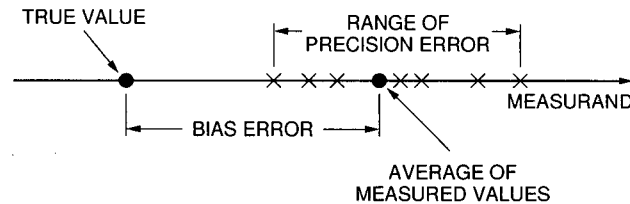


Figure 2.4: Distinction between bias error and precision error.

Precision errors can originate from the sensor itself, the industrial system, or from the environment. They are usually caused by uncontrolled variables in the sensing process.

Uncertainty is an estimate (with some level of confidence) of the limits of error in the measurement. The degree of uncertainty may be reduced through the use of calibrated, high-quality sensors. *Accuracy* is a term commonly used to specify uncertainty. It is a measure of how closely a measured value agrees with the true value. *Precision* is used to characterize the precision error of a sensor. In general, the accuracy of a sensor cannot be any better than the measurement constraints provided by the sensor precision, and often, is much worse.

Accuracy is often degraded by hysteresis errors (bias), resolution errors (precision), repeatability errors (precision), linearity errors (bias), zero errors (bias), sensitivity errors (bias), and drift and thermal stability errors (precision), among others.

Digital signal processing requires the conversion of analog sensor signals into digital form. A/D converters are used for this purpose; however, they are prone to three bias errors: linearity, zero, and sensitivity (or gain) errors. Since the output of an A/D converter changes in discrete steps, there is also a resolution error (uncertainty) known as a *quantizing error*, which is a type of precision error. Together, these errors are known as *elemental error sources*.

To facilitate the identification and comparison of sensing errors, ASME/ANSI suggests grouping elemental errors into three categories: calibration errors, data acquisition errors, and data reduction errors [47]. *Calibration errors* originate in the calibration process and may be caused by uncertainty

in standards, uncertainty in the calibration process, and randomness in the calibration process. Hysteresis and non-linearities are usually included here. *Data acquisition errors* are introduced into the measurement when the sensor is making a specific measurement. These include random variation in the measurand, loading errors and A/D conversion errors. *Data reduction errors* are caused by a variety of errors and approximations used in the data reduction process.

Grading and inspection tasks rely upon various sensors to obtain information about the objects under consideration. Accurate decisions require that the sensed information be valid and robust. Validation of data through sensor integration provides one mechanism by which uncertainty may be represented and collaboratively reduced. A multisensor integration system must check for errors which are the result of unexpected events, such as sensor malfunctions or environmental changes, which cause a device to fail to perform within specifications. If found, an attempt must be made to correct the cause of the error. This is usually handled through an exception and error handling mechanism.

2.8 Object Modelling

To utilize a multisensor architecture for object grading, a model of the object is required. An object model is necessary for a computer system to perform object recognition. The model provides a generalized description of each object to be recognized. The model is used for tasks such as accurately determining object boundaries in an image and choosing an object's best class membership from among many possibilities. For industrial grading applications, the object model must represent the important features which designate the 'grade' or value of a particular object. Ideally, the model is simple to construct.

Methodologies for object recognition and representation abound; however, much of the research in the field has focused on the recognition of generic objects, categorizing objects into broad groupings [50]. Many of these are further limited by requiring geometric representations of the objects [51,52]. With the exception of facial and handwriting recognition [53–55], little work has been done to develop systems capable of detecting subtle differences. This is the requirement of an industrial inspection and grading system where objects are classified on the basis of subtle differences. The problem is not one of differentiating an apple from an orange, but rather one of discriminating

the quality of a particular apple based on such cues as colour, size, weight, surface texture, and shape. Despite this, there are a number of object models which have been developed which are applicable, at least in part, to the product classification problem.

Studies into how humans perform object recognition have yielded some interesting results. Biederman [56] has suggested that objects are recognized, and may therefore be represented, by a small number of simple components and the relations between them. These simple geometric components are called *geons* (for geometrical ions). Objects are typically segmented at regions of sharp concavity. Geons and relations among them are identified through the principle of non-accidentalness. In other words, critical information is usually represented by nonaccidental properties — an accident in viewpoint should not affect the interpretation. These basic phenomena of object recognition indicate the following:

1. The representation of an object should not be dependent on absolute judgments of quantitative detail.
2. Information which forms the basis of recognition should be relatively invariant with respect to orientation and modest degradation.
3. A match should be achievable for occluded, partial, or new exemplars of a category.

These ideas form the basis for the theory of recognition-by-components (RBC). The associated stages of processing are presented in Figure 2.5. This indicates that for feature-based recognition distinguishing features are used to recognize and differentiate objects. This method is efficient, as it is not necessary to discriminate every feature of every object. By closely modelling the object representation to the human methodology, this scheme may also have the advantage of being more intuitive to the user.

An interesting parallel may be drawn from this to the series of steps that a typical vision-based grading system follows in recognizing and classifying the objects in a given image, as illustrated by Figure 2.6.

Havaldar, Medioni, and Stein [57] have developed a system for generic recognition based on Biederman's ideas. Images are processed to extract edge sets from which features of parallelism, symmetry, and closure are identified. These features are then grouped and represented within

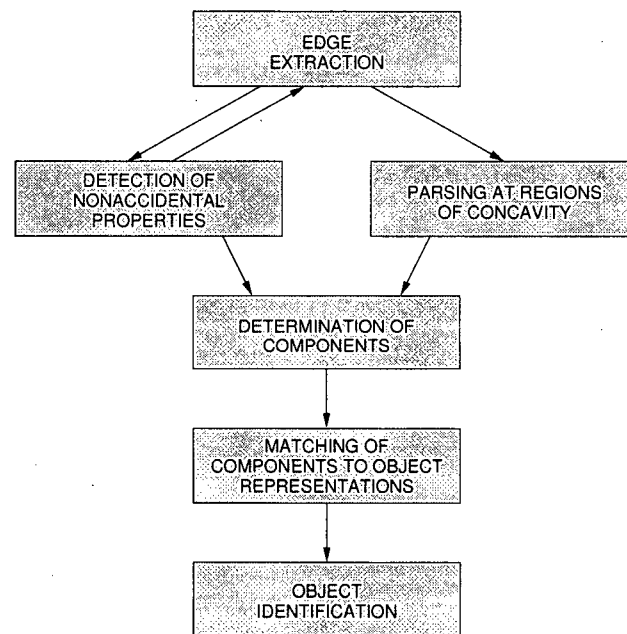


Figure 2.5: Presumed processing stages in human object recognition [56].

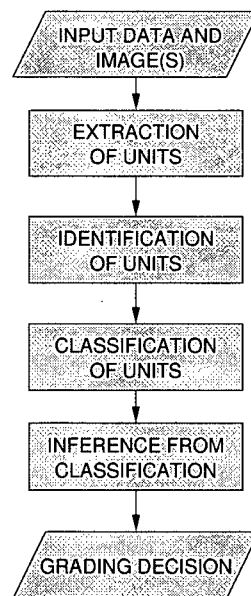


Figure 2.6: Four steps in object grading.

an adjacency matrix. This is a robust system, able to recognize objects which deviate from the exemplar; however, it is not designed to recognize the deviations themselves — a requirement for object classification.

A feature-based object model was developed by Tomita and Tsuji [58] for object recognition from texture features. Their primary application was a system designed to recognize various structures of the human brain visible in computed tomography (CT) images.

Objects are represented by a connected graph structure as shown in Figure 2.7. Each node represents a kind of object to be recognized in the image; the root node represents a category of image. The node contains slots for the name, the type of unit in the image, and the properties of the unit. Nodes which are white indicate that the object is always recognized; black nodes signify that the object may not always be present, as in the case of abnormalities. Solid links are used to represent a parent-child relationship between nodes. Dotted links represent an *OR* relationship — only one of the linked objects will be recognized. This relationship may be used to represent an object which, due to possible variations, cannot be defined by a single node.

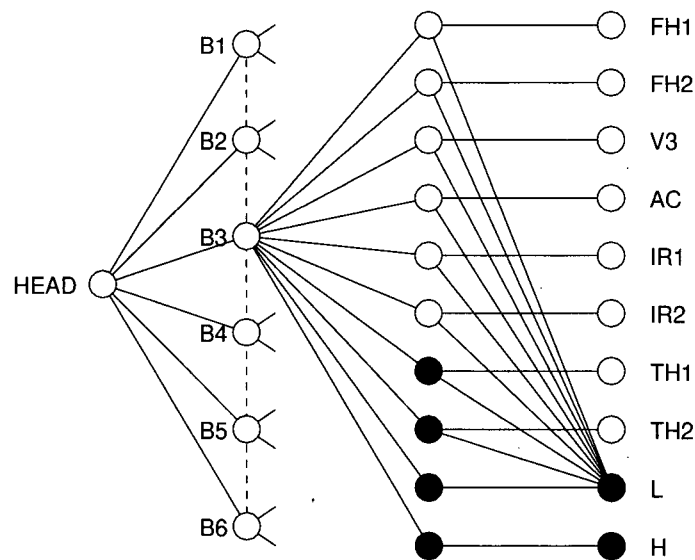


Figure 2.7: Model used to recognize cranial CT images [58]. White nodes indicate brain features that are always present; black nodes represent abnormal features.

Models are built in an interactive manner. Programs are selected and applied to input images to extract the desired features. Parameters are adjusted until the desired results are obtained. Successfully extracted units are identified to the system. Each unit generates a new node in the graph; each unit may be further subdivided into smaller units. Once the initial model has been constructed, the model may be refined by adjusting the program parameters, adjusting the object

properties and/or relations, declaring an *OR* relationship between objects, or by specifying that an object may not always be present.

Other feature-based systems include the work of Han, Yoon, and Kang [40] who identify a number of features for automatic surface inspection. Lang and Seitz [59] represent and recognize objects through the use of a number of hierarchical feature templates.

Fuzzy logic has been used by a number of researchers to describe varying relationships between features. Cho and Bae [60] describe objects in terms of functional primitives which are constructed from extracted shape primitives. An object is represented by a collection of these primitives related by fuzzy memberships. Luo and Wu [54] and Lee and Huang [55] have developed methodologies for handwritten Chinese character recognition. In these systems, each stroke is extracted from the character as a feature. Features are then classified as particular stroke types, each with an associated degree of fuzziness. The classified features are then combined based on connectedness and regularity to arrive at a predefined character classification.

While none of these approaches are directly applicable to representation of non-uniform products for the purpose of classification, each presents some interesting ideas for the basis of such an object model. A feature-based system will allow for the efficient representation of the distinguishing characteristics of objects to be classified. Fuzzy logic provides a mechanism by which human expertise may be applied in a form very close to our natural language [61]. Relating object features with fuzzy membership functions should enable the system to incorporate human expertise for the determination of object classifications.

Chapter 3

Object Modelling

3.1 Introduction

An intelligent system which attempts to perform object recognition must have a facility for perception. Machine perception consists of converting raw sensor information into a form which may be utilized within the system to accomplish a task. To facilitate this conversion, an object model is used as the interface between the real environment and the internal processes which are dependent on the external information. The object of interest is represented by the object model through characteristic properties and relationships between features, with a particular focus on those features which are most relevant to the application. Therefore, an object model is a generalized description of each object to be recognized.

3.2 Rationale

As discussed in Section 2.6, demand for improved automated quality assurance systems has led to the development of a number of vision-based multisensor systems. Typically, these systems are unstructured, complex, and difficult to maintain and modify. To enable industrial users to better react to changing market conditions and improved technology, a formal approach to system design is needed to replace these ad-hoc systems.

In this work, the Extended Logical Sensor Architecture (ELSA) has been developed to address a number of these limitations in current industrial practice. The purpose of this architecture is to

provide a structured, yet flexible methodology for building robust sensor systems aimed at product inspection. A well-defined, structured object model is the starting point of this organized approach to the design and construction of a multisensor integration system.

There are two objectives that determine the structure of the object model used within ELSA. The first objective is to provide a representation for objects which exhibit deviations from an ideal template or model, or an object for which an ideal cannot even be concretely established. The model should allow for the representation of both quantitative and qualitative information. This addresses a problem of particular relevance to non-uniform product inspection and grading. The structure of the model should provide users with an intuitive understanding of how to construct and represent real-world objects.

The second objective is to develop the object model as a guide for the selection of components and construction of an ELSA system. The features represented in the object model should guide the selection of the sensing devices and/or processing algorithms required to extract them. The high-level representations of the object and its classifications should provide a basis for inferring the proper identity of the object from the extracted features.

The object model then serves two purposes: (i), In the completed system, the object model is used to recognize and represent objects that are presented to the system sensors; (ii), once defined, it may be used to specify the components that are necessary for the system to identify and classify objects.

3.3 Approach to Modelling

There are two approaches which may be taken towards object modelling for classification and grading. They differ in the how the object is represented and therefore how it is identified.

The traditional approach to object recognition attempts to identify an entire object based upon the features contained within the object model [50,62]. Once an object has been identified, extracted object properties may then be used for further evaluation based on attributes such as size, colour, and mass. Recognition proceeds in a top down manner from the root nodes of the model graph, which represent the different objects or object classifications. The selection of a particular parent is contingent on the successful identification of all descendant objects. Should the system fail to

find an expected object at a particular level, the system returns to the previous level and attempts to follow another branch. If a proper match cannot be found, the system issues an error message requiring the user to improve the object model.

The second approach defines object models somewhat differently. Instead of attempting to identify an object based on the discrimination of every feature of the object, only distinguishing or characteristic features are extracted. These features are then combined to produce object classifications. The presence or absence of particular features and the associated object properties may then be used to classify the object into a particular grade. This idea is supported by the theory of recognition-by-components (RBC) [56], which suggests that objects may be represented by a small number of simple components.

It is this feature-based approach that is adopted herein. Unlike the first approach which is best suited to simple objects, it is applicable to both simple and complex models. Objects that demonstrate deviation from an ideal model may be represented using appropriate features combined into classifications. Additionally, by identifying only those features necessary for object recognition and/or classification, the storage requirements for object representation are reduced. Concentration on distinguishing features also reduces the processing requirements for the extraction of features from the environment.

3.4 Model Structure

In the ELSA object model, objects are represented by a connected graph structure similar to that proposed by Tomita and Tsuji [58]. The components of the structure are shown in Figure 3.1. This is a top-down representation of an object, consisting of a number of layers of abstraction. Object nodes are used to represent salient features of an object. The object itself is represented at the highest level of abstraction within the classification layer. Below this lie nodes representing the high-level features upon which classifications are made. Traversing down the graph, further into the feature layer, other nodes represent the mid and low-level features of the object. Each subsequent level becomes more and more detailed. This enables compact and efficient object models. Only the level of detail required for identification or classification need be specified.

This approach allows for scalable complexity of the object model. By adding nodes and layers

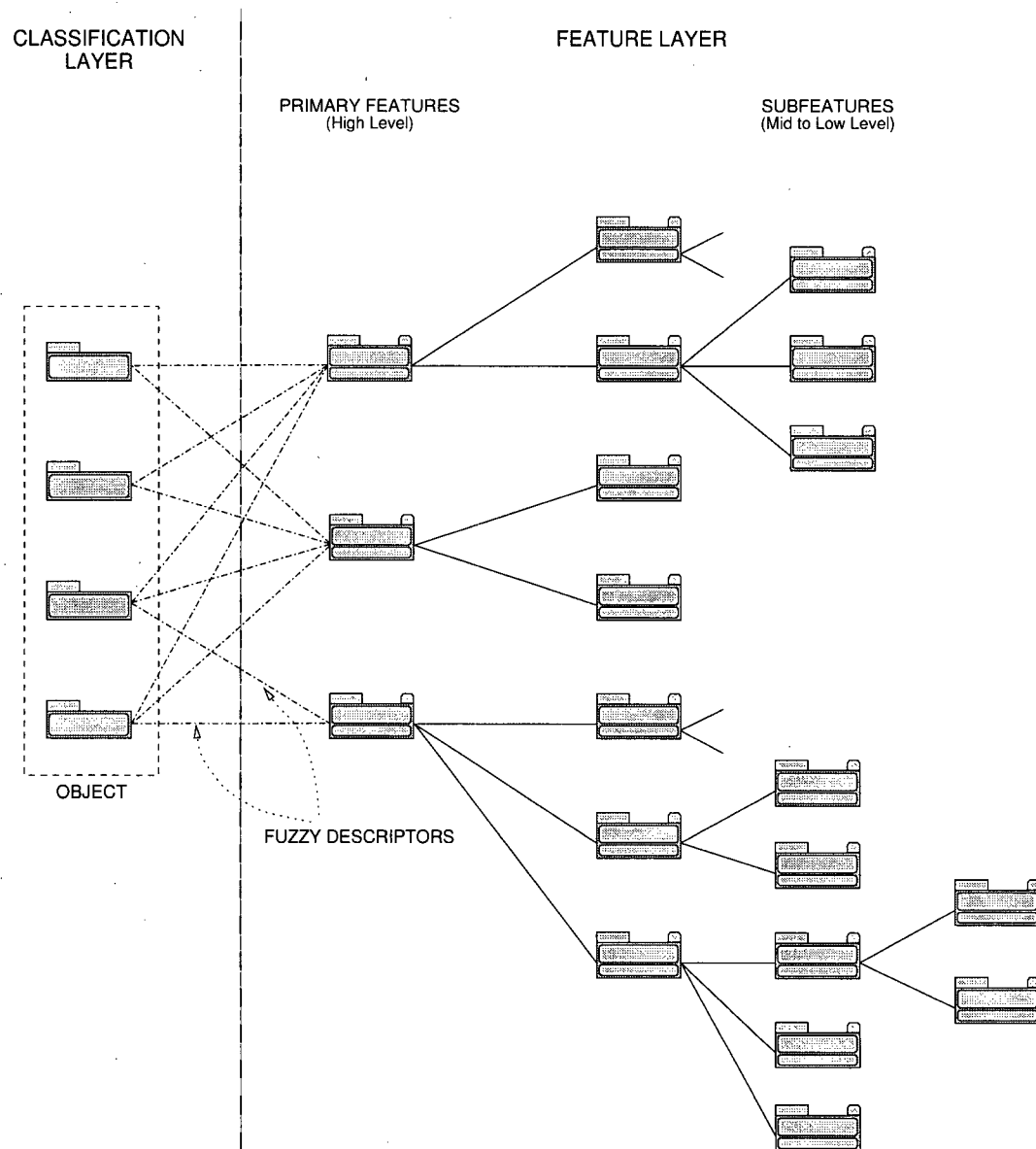


Figure 3.1: Graph structure for object representation.

to the graph, models may be made as simple or complex as required to properly model the objects considered by the system. The hierarchical structure minimizes the disturbance to the model should a feature used for classification require modification. Thus, refinement may focus on specific features and classifications without disturbing other classifications.

3.4.1 Classification Layer

The classification layer represents the kind (grade, grouping, category) of the object. Different object classifications may be grouped within the classification layer because they each share similar features or qualities. This is the principle advantage of feature-based object recognition. The features common to each object need not be specified. Rather, the features that distinguish one object from another are used. For example, a classification layer could represent apples; different classifications could include ripe, bruised, large, and small. The common features describing the general characteristics of all apples: stem, skin, shape, etc., need not be articulated.

Each classification is defined by associating it with the appropriate primary features. Associations are made using fuzzy links, which are described in Section 3.6.3. An object whose relevant features are invariant or which does not require classification may be defined with a single node in the classification layer.

3.4.2 Feature Layer

A feature is defined as a distinct quality, detail, characteristic, or part of an object. An object may be described and recognized as a collection of features. The ELSA object model categorizes features based on the level of abstraction. The highest-level features are termed *primary* features. These features are linked directly to the classification layer and serve to define each classification.

Most primary features are themselves composed of one or more *subfeatures*. Subfeatures represent lower-level, less abstract features. As the graph is traversed downward, features become more specific and detailed. At the extreme, the lowest-level subfeatures are called *atomic* features. These represent features that are indivisible. The unprocessed data from a sensor is often represented as an atomic feature. The nodes of the feature layer are connected with unconditional links.

3.5 Properties of Objects

Within the data representation, objects may have two different types of properties, namely: physical object properties and relational properties. Relational properties are dependent upon the extraction of a pair of physical properties which are then related in some way. Due to this increased complexity, objects are modelled using only physical object properties whenever possible.

3.5.1 Physical Properties

Physical object properties are used to describe intrinsic qualities of an object. Each property is characterized such that it may be considered independently from any others. Examples of physical object properties include position, mass, temperature, shape, colour, intensity, and texture.

These properties are represented within the model structure with the appropriate data structure. For example, colour may be represented at a low level with a data structure containing the RGB (red, green, blue) or HSI (hue, saturation, intensity) channel values. Abstractions may occur such that the degree of a particular colour value is interpreted from the HSI data. Such a data structure could indicate the hue, e.g. RED, and a value that specifies the 'redness' of the object. This value could be a measure in the range [0–1]: 0 representing no presence of red; 1 complete red saturation. Similar structures would be defined for other types of physical properties.

3.5.2 Relational Properties

Relational properties describe an object in relation to other objects. Unlike physical object properties, each relational property is dependent upon at least one other object. Symmetry, adjacency, relative position, and relative orientation are examples of relational properties.

Whereas physical properties are computed for each feature extracted, it is unlikely that all of the possible relations between each pair of objects can be computed, even for a small number of objects. This is due to the large number of relations which may be defined. Therefore, only those relations which are specifically identified by the user are computed. A relation between objects is defined only when the system is unable to recognize objects based on the physical properties of the objects themselves.

Relational properties are represented within the structure of the object model using a data

structure which contains a field for each object, a field to identify the type of relation, and a field for parameters which specify exactly how the objects are related.

3.6 Model Components

The object model is comprised of a number of different components. Object nodes are used to represent object features. Subfeature dependencies are represented using unconditional links; object classifications are specified using fuzzy links. The following subsections provide details about each component. Implementation issues are discussed in Appendix A.

3.6.1 Object Nodes

Each node of the graph represents a recognizable object or feature. An object may refer to the representation of any signal, attribute, or thing which may be recognized by the system. These may be complex features extracted from information provided by one or more sensors. Each node may be a parent node, that is, it is associated with one or more child nodes which further detail features of the parent node. In other words, the child nodes are representative of the subfeatures of the parent node. For example, a parent node may be the size of an apple, while child nodes may include the volume, area, and height of the apple. Alternatively, a node may contain simple crisp measurements provided by a single sensor, for example, mass and temperature. Primary features are represented by root nodes that, by definition, do not have a parent. The components which comprise the object node are outlined in Table 3.1.

Table 3.1: Components of object node for feature representation.

Component	Description
Object Name	Uniquely identifies the object or feature.
Object Type	Indicates the type information that this particular node represents.
Physical Properties	Data structure for the physical properties of the feature.
Relational Properties	Data structure for the relational properties of the feature, if required.
Free Tag	If set, it indicates that this feature may not always be present.

The node structure contains the name of the object, the type of object, and the object properties. Nodes that represent features which are not always present are marked by a *free node tag*. This usually applies to features that correspond to object classifications that are defective or otherwise deviate from the ideal. Links to parent and child nodes are maintained within the structure. This is illustrated in Figure 3.2.

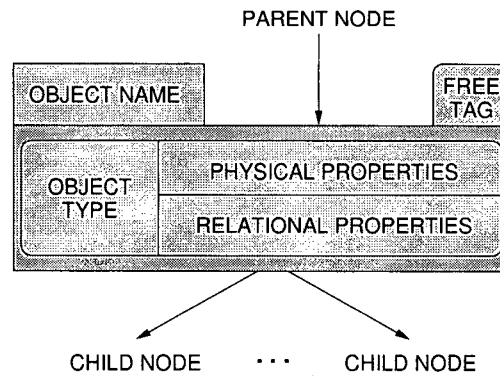


Figure 3.2: Object node for feature representation.

Classification nodes, Figure 3.3, may be considered as a special case of an object node. They do not have parents, do not maintain object properties, and do not have free node tags. Instead, the primary features upon which the classification is dependent are stored along with the corresponding fuzzy feature descriptions.

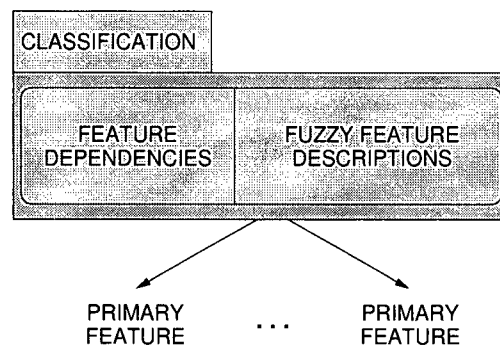


Figure 3.3: Classification node.

3.6.2 Unconditional Links

Unconditional links are used to represent parent-child relationships between features. They are unconditional in that the relationship between the nodes (which correspond to features) is constant and is not modified in any way. Unconditional links are stored within the nodes as pointers. Graphically, they are represented as a solid line.

3.6.3 Fuzzy Links

Similar to unconditional links, fuzzy links represent a relationship between object classifications and primary features (root object nodes). They differ by attaching additional information in the form of a fuzzy descriptor. The fuzzy descriptors are used by the classification nodes to help assess how the primary features contribute to the representation of the object. Fuzzy descriptors are realized using fuzzy logic membership functions.

Fuzzy logic provides a mechanism by which human expertise may be applied in a form very close to our natural language [61]. This enables the system to incorporate human expertise relating features to the determination of object classifications. This is especially useful for applications such as non-uniform product grading that tend to use subjective interpretations of product quality. For example, the ripeness of an apple may be described using linguistic variables such as *not very red*, *sort of green*, and *slightly red* as opposed to some quantification of apple colour in RGB or HSI colour space. Such descriptors may be constructed from a number of atomic terms as discussed by Zadeh [63].

3.6.3.1 Linguistic variables

Linguistic variables are in the form of natural language phrases. They are used to label fuzzy subsets from the universe of discourse, U . A linguistic variable x , over the universe $U = [1, 100]$ of *weight*, may have values such as: *light*, *not light*, *very light*, *not very light*, *heavy*, *not very heavy*, *not light and not heavy*, etc.

In general, the value of a linguistic variable is a composite term $x = x_1x_2 \cdots x_n$. In other words, x is a concatenation of atomic terms x_1, \cdots, x_n . There are four categories of atomic terms:

1. *Primary terms*, are labels of specified fuzzy subsets of the universe of discourse. (e.g. *light* and *heavy*).
2. The negation *not* and the connectives *and* and *or*.
3. *Hedges*, such as *very*, *much*, *slightly*, *more or less*, etc.
4. *Markers* such as parentheses.

Hedges are used to generate a larger set of values for a linguistic variable from a small collection of primary terms. Hedges allow definition of subsets while maintaining a minimum set of primary terms. They are particularly useful for translating human descriptions into mathematical notation. The hedge h may be regarded as an operator. h transforms fuzzy set $M(u)$ into the fuzzy set $M(hu)$. These form the foundation for information granulation and computing with words.

For example, consider the hedge *definitely* which acts as an intensifier. This hedge may be implemented as a concentration operation. Like all hedges, it generates a subset of the set upon which it operates. Therefore, *definitely* x , where x is a term, may be defined as:

$$\text{definitely } x \triangleq x^2 \quad (3.1)$$

or, more explicitly:

$$\text{definitely } x \triangleq \int_U \mu_x^2(y)/y \quad (3.2)$$

This is further illustrated by the following equations, plotted in Figure 3.4.

$$\begin{aligned} x &= \text{heavy object} \\ &\triangleq \int_{50}^{100} \left(1 + \left(\frac{y-50}{5} \right)^{-2} \right)^{-1} / y \end{aligned} \quad (3.3)$$

$$\begin{aligned}
 x^2 &= \text{definitely heavy object} \\
 &\triangleq \int_{50}^{100} \left(1 + \left(\frac{y-50}{5} \right)^{-2} \right)^{-2} / y
 \end{aligned}
 \tag{3.4}$$

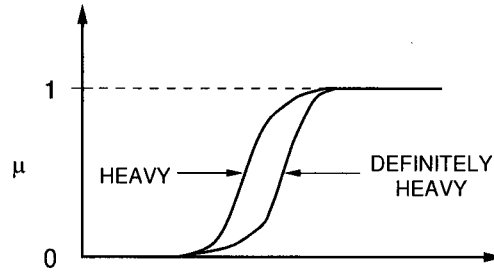


Figure 3.4: Effect of hedge *definitely*.

Linguistic variables constructed from these atomic terms are used to describe how primary features relate to object classifications. A minimum set of primary terms is chosen for a given feature or classification. In most cases, this will be a pair of descriptors such as: cold/hot, young/old, light/dark, small/large. Additional classifications are achieved through the use of negation, connectives, and hedges.

3.6.3.2 Membership functions

Linguistic variables are associated with fuzzy membership functions. These membership functions, referred to by the linguistic variable, are used to define the fuzzy descriptors used to construct object classifications.

Many features, such as shape and texture, are not easily quantified. To enable the classification of such features, the membership functions *no*, *low*, and *high* are used to express the confidence in the detection of the feature. These may also be thought of as describing a feature as *does not* belong to the class, *could* belong to the class, and (*definitely*) *does* belong to the class. As shown in Figure 3.5, these functions span the universe 0 to 1. This is intended to provide users with an intuitive feel for the specification of classifications. The user does not consider values or fuzzy membership, but rather the linguistic variables *no*, *low*, and *high*.

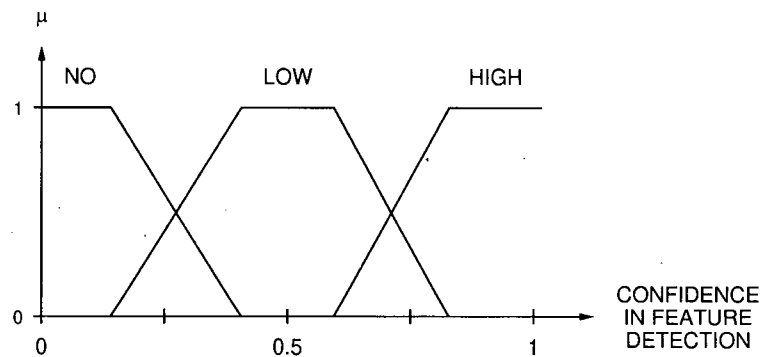


Figure 3.5: Membership function used to represent confidence in the detection of a particular feature.

For features that are easily quantified, such as length and mass, the universe of discourse (range of expected values) is specified along with linguistic variables for the classifications in this universe. Triangular or trapezoidal membership functions centred at the mean values of each variable are used, since with sufficient representation the membership function shape is not critical [64]. The choice to use trapezoidal membership functions is based on the need to encompass a broad range of values by a single fuzzy label. Most often this is at the limits of the universe of discourse, but may also be used to specify narrow overlapping regions between labels while using a minimum number of labels to cover the universe of discourse.

3.7 Model Definition

The object model is defined by first identifying the primary features. Each is associated with an object node which occupies the top of the feature layer. If necessary, each primary feature is decomposed into subfeatures — each represented by an object node. These are linked together using unconditional links.

The definition of the classification layer follows. Object classifications are associated with classification nodes. These are then linked to appropriate primary feature nodes using fuzzy links. Each fuzzy link is assigned a fuzzy descriptor which describes how the feature is used to represent the classification. The detailed algorithm used for the construction and refinement of the object model is presented in Chapter 5.

3.8 Summary

In this chapter, the object model used by the architecture has been presented. This structure satisfies two objectives. The first is to provide a representation for features and objects which allows for the quantification of deviations from an ideal model. Secondly, it provides a structure by which the user may easily understand how objects are modelled while guiding the selection of sensing devices and the development of the inference engine. These components are presented as part of ELSA in the following chapter.

Chapter 4

System Architecture

4.1 Introduction

This chapter presents the basic structure and functions of the Extended Logical Sensor Architecture for multisensor integration. A system designed using the principles of ELSA is composed of a number of different modules. The primary modules are the logical sensors and inference engine. Other modules — such as those for integration, validation, and diagnostics — provide vital, though secondary, support to the operation of the system.

The definition and construction of an ELSA-based multisensor system is based on the object model outlined in Chapter 3. The feature layer guides the selection and interaction of sensor components. The classification layer is used to construct a rulebase which defines how the sensor information is used and what the system can infer from it.

The relationship between the object model and the system architecture allows the system to be designed with inherent modularity and scalability. Additionally, by utilizing a standard approach, components may be shared and reused by applications with differing object models and logical sensor hierarchies. Examples of the construction of an ELSA system are given in Chapter 6.

The ELSA architecture may be decomposed into three groups, according to the following tasks:

1. **Sensing:** The acquisition of information from the environment which is used as the basis for inference and decision making.
2. **Inference:** The combination of the sensory information with information contained in a

knowledge base to infer decisions.

3. **Action:** The conversion of decisions into commands and signals which control process machinery.

The structure of ELSA is illustrated in Figure 4.1. An object-oriented approach to the system configuration has been adopted. The encapsulation of the primary components leads to a scalable and flexible system which is particularly suited to industrial grading tasks. The system may be easily reconfigured to adapt to advances in sensor and processing technologies or changing market demands. Due to the nature of industrial inspection and grading, the primary focus of this work is on the sensing and inference groups.

Sensing is performed by the coordinated actions of the sensors, the Integration Controller, and the Validation and Diagnostic modules. Sensors are encapsulated by a logical sensor model. The Integration Controller is capable of coordinating the reconfiguration of the sensor hierarchy to meet process goals. This is assisted knowledge by contained in the Knowledge Base which is shared with the Inference Engine.

Process decisions are made by the Inference Engine. The validated sensor information from the sensing group provides the required input to the Rulebase. The action group includes the Post Processor, drivers, and process machinery. Control systems for grading systems typically range from very simple to extremely complex. Herein, the details of the control issues associated with the action group are not considered and are open problems for future work.

4.2 Logical Sensors

The logical sensor hierarchy structures data in a bottom-up manner. The raw data collected by the physical sensors is processed through different levels of logical sensors to produce high-level representations of sensed objects and features. This approach offers considerable flexibility. High-level tasks may be implemented without regard to the specific sensing devices. The low-level physical sensors and low-level data processing routines are invisible to the higher levels. That is, to higher-level sensors, each antecedent logical sensor appears as a single entity with a single output, regardless of the scope of *its* antecedents. Using the logical sensor model, a hierarchy of subordinate and

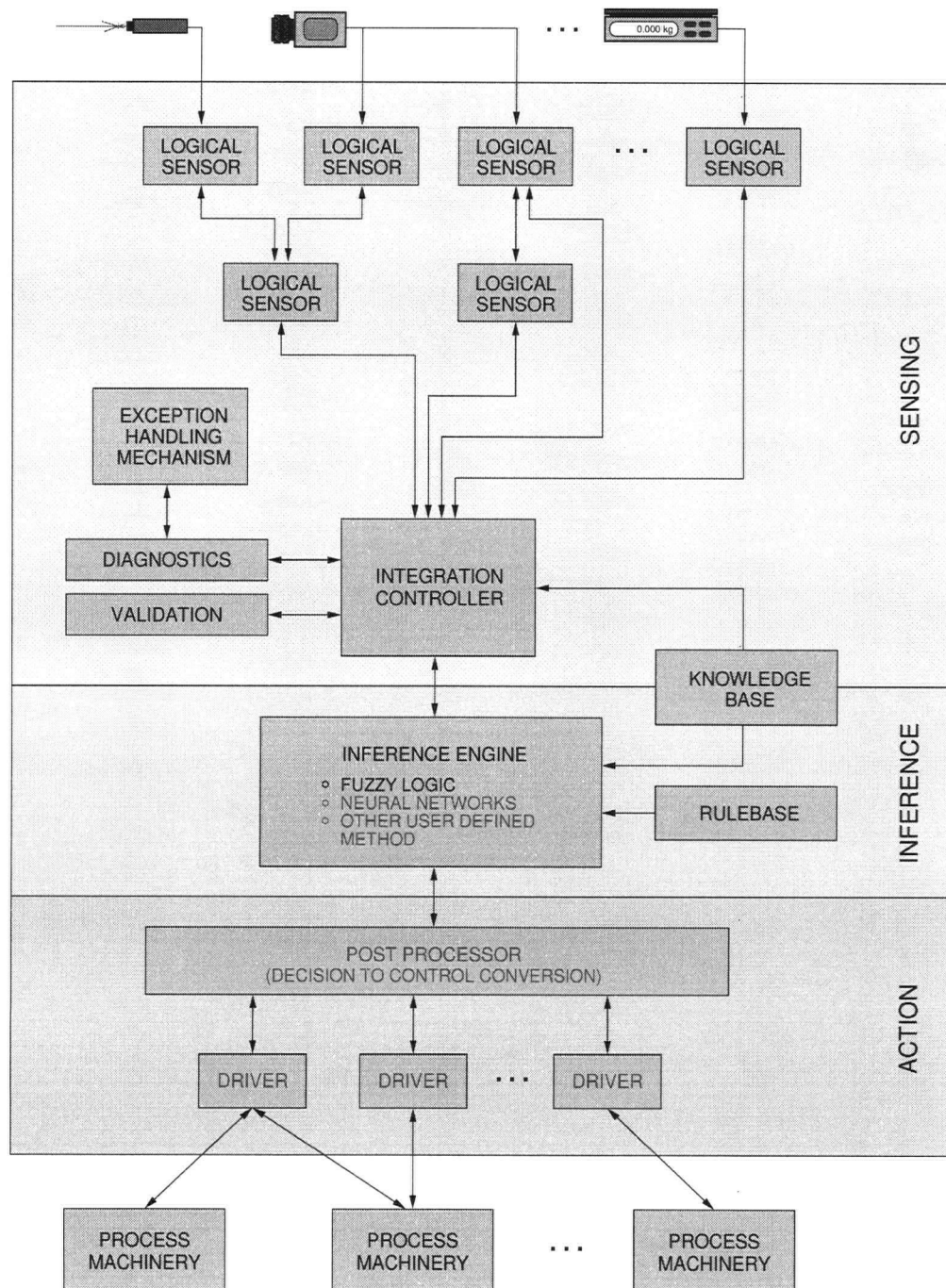


Figure 4.1: Overview of Extended Logical Sensor Architecture.

controlling sensors can be built, ultimately providing sensor input to the Integration Controller.

The logical sensor model outlined in Section 2.3.1 has been extended herein for a model-driven open architecture. As shown in Figure 4.2, the proposed Extended Logical Sensor (ELS) is comprised of a number of different components. The components are object-oriented by design; each component is responsible for a single task within the sensor. A list of these components and tasks is given in Table 4.1. As indicated in the table, a few components are unchanged (U) from the original logical sensor specification [20]; others are based on extensions (E) to the specification [21, 26, 27]; and the balance are novel (N) in this work. The ELS strongly encapsulates the internal workings of each logical sensor while allowing the modification of the sensor's operating characteristics. Most of the components of this revised model are outlined in greater detail in the sections referred to in the final column of Table 4.1.

The control command mechanism is flexible enough to allow active sensors; for example, a camera in an active vision system may be repositioned to bring an object of interest into (better) view. However, since the target applications are industrial in nature, namely, inspection and grading tasks, herein the sensors are assumed to be passive.

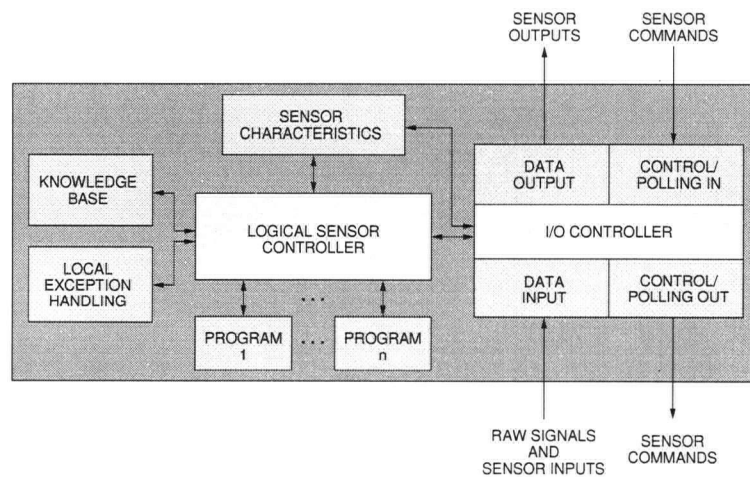


Figure 4.2: Basic components of an Extended Logical Sensor.

As will become apparent, the implementation of an ELS requires an understanding of signal processing. This is knowledge that most industrial users will not possess. They will understand *what* they would like the ELS to do, but not necessarily *how* to accomplish it. This limitation is

Table 4.1: Summary of Extended Logical Sensor components.

Component Group	Component	Description	Origin ^a	Reference
Sensor Characteristics	Logical Sensor Name	Uniquely identifies a particular logical sensor to the system. By definition, a name may not be duplicated within the hierarchy. Similar sensors are numbered consecutively.	U	Henderson and Shilcrat [20].
	Characteristic Output Vector	A vector of types which serves to define the output vectors that will be produced by the logical sensor.	U	Henderson and Shilcrat [20].
	Sensor Function	A description of the functionality that this sensor provides. Provided in human readable form.	N	Section 4.2.1
	Sensor Dependency List	A list of dependencies for the logical sensor, accounting for each logical sensor that serves as input to the contained programs.	N	Section 4.2.1
I/O	I/O Controller	Monitors, redirects, and packages data and control commands for inter-sensor communication.	E	Section 4.2.2.1; and, Henderson et al. [21]
	Data Input	Consists of signals from transducers and data from logical sensors.	N	Section 4.2.2.2.
	Data Output	Output in the form of the characteristic output vector, error messages, or polling results.	N	Section 4.2.2.3.
	Control Input	Interprets the control structure used for commanding and adjusting sensors for changing conditions and goals.	E	Section 4.2.2.4; and, Dekhil and Henderson [27].
	Control Output	Control commands to subordinate sensors. May be generated by sensor or passed through from higher level sensors.	U	Section 4.2.2.4.
Controller	Logical Sensor Controller	Acts as a "micro" expert system to ensure the optimal performance of the logical sensor.	E	Section 4.2.3.1; and, Henderson and Shilcrat [20].
	Local Exception Handling	Internal diagnostics and error handling. Works in conjunction with logical sensor controller. Attempts to classify the error and then rectify the problem using a predefined recovery scheme.	E	Section 4.2.3.2; and, Dekhil and Henderson [26, 27].
	Local Knowledge Base	Contains information on interpretation of control commands for adjustment of parameters and selection of programs. Also stores default parameters used during initialization and reset.	N	Section 4.2.3.3.
Programs	Device Drivers	Used to interpret raw signals from physical sensory devices.	E	Section 4.2.4.1.
	Processing Algorithms	Signal processing routines used to extract features and information from sensor data.	U	Henderson and Shilcrat [20].

^aU – unchanged, E – extended, N – novel.

overcome to some degree by the development and provision of an ELS library which contains a variety logical sensors for many common signal processing operations. When a required ELS is not available in the library, it will be necessary to have others implement the ELS.

For these developers, an ELS base class is provided which serves as a template for the design of Extended Logical Sensors. The ELS model is implemented as a C++ class library following the principles of object-oriented software design. Individual sensors inherit the basic structure and common functionality. Customizations are achieved either by overriding base classes and functions or providing new ones where necessary.

The subsections that follow outline the major components of an ELS. The ELS base class is outlined in Appendix B.

4.2.1 Logical Sensor Characteristics

The logical sensor characteristics refer to a set of properties specific to each logical sensor (LS). This information is publicly accessible, enabling other logical sensors, or the Integration Controller, to poll the sensor and determine the sensor's identity and capabilities. The components which comprise the logical sensor characteristics are: the Logical Sensor Name, the Characteristic Output Vector, the Sensor Function, and the Sensor Dependency List. The first two characteristics were defined by Henderson and Shilcrat [20]; the other characteristics are new, and are described below.

The Sensor Function provides a description of functionality of the logical sensor. This description is in human readable form so that a user may effectively browse through a library of logical sensors. As an example, a Canny edge detection ELS [65], would have a description indicating that it was capable of identifying sets of edge pixels from a two-dimensional array of pixel intensity values. In addition, comments on accuracy and computational complexity (speed and memory requirements) would assist the user and the system in comparing this edge detector with others which may be available. This information may then be used to select the most appropriate edge detector for a given task.

The Sensor Dependency List provides a list of the logical sensors subordinate to the ELS being polled. Each ELS which provides input to one of the logical sensor programs is considered as a subordinate. An ELS is identified by its Logical Sensor Name. This list is automatically generated

as the ELS hierarchy is constructed.

4.2.2 I/O

4.2.2.1 I/O Controller

The I/O Controller is an extension of the Control Command Interpreter [21], that provides a specification for control to the original logical sensor specification [20]. The I/O Controller oversees all inputs and outputs from the LS and monitors, redirects, and packages data and control commands for inter-sensor communication. For control commands, the controller works as a pass-through buffer. The destination logical sensor name of each control object received by the I/O Controller is first checked to determine if the command is intended for the particular sensor. If so, the control command is interpreted and sent to the LS Controller for processing; if not, it is passed through to lower-level (subordinate) sensors.

One can note that, higher-level sensors may only be aware of the *function* of each subordinate ELS. The details of the actual algorithms — and in the case of sensors with multiple programs, the currently selected algorithm — is hidden from higher-level sensors by encapsulation. As a result, commands (and associated parameters) generally request a desired *effect*. For example, a command to increase the number of edges extracted from an array of pixel intensities would be of the form INCREASE EDGES. The specific algorithm used need not be known. This command would be passed down through the hierarchy to the edge detecting ELS. At this sensor, the controller, Section 4.2.3.1, would interpret this command and, drawing upon information contained in the Local Knowledge Base, adjust specific algorithm parameters accordingly (such as reducing mask size or threshold values).

A number of control commands are defined for all logical sensors, namely, commands used for sensor initialization, calibration, requests for sensing, testing, and reconfiguration. A complete list of standard commands is provided in Table 4.2. For example, the polling command is used to query lower-level sensors about the logical sensor characteristics described in Section 4.2.1. The applications of other standard commands are outlined in Section 4.2.3.1.

4.2.2.2 Data Input

The data sources for an ELS may take two forms:

1. **Raw signals from (physical) transducers:** Signals from digital devices are input directly to a software driver. Analog signals are first converted into a digital form using an A/D converter.
2. **Data from logical sensors:** As will be discussed in Section 4.2.2.3, logical sensor data is packaged in the form of the Characteristic Output Vector (COV). These output vectors serve as the sensor inputs for higher-level sensors. This data is then used as input to the processing algorithm(s) of the logical sensor.

To properly interpret data from subordinate sensors, the I/O Controller must have an internal copy of the characteristic output vector for each connected lower-level ELS. This internal copy is obtained through sensor polling.

4.2.2.3 Data Output

The data output module serves to package the ELS output into one of three forms, as outlined below:

1. **Output vector:** The data output module serves to package the data from a logical sensor program into the form of the COV. This enables the sensor to pass a data package, without identifying each component.
2. **Error message:** Failure of an ELS may occur due to the failure of a lower-level LS or an inadequacy of a contained algorithm. In either case, the confidence measure which accompanies each ELS output will fall below a specified tolerance. An error message will then be passed in place of the output vector.

The confidence measure is generated by the ELS. In the case of an encapsulated physical sensor, the uncertainty measure is based upon the specifications and/or known operational characteristics of the device. Algorithms within the ELS must provide routines which calculate the uncertainty associated with each output value. Confidence is represented as a real-valued

number in the range: $0 < c < 1$. A measure near 0 indicates little confidence in the result; while a measure near 1 indicates a high level of confidence in the sensor output.

3. **Polling result:** This consists of information obtained from the logical sensor characteristics in response to a query from the Integration Controller or a high-level logical sensor.

4.2.2.4 Control Input

The logical sensor model provides a control structure which allows for the adjustment of logical sensors in response to changing conditions. Possible adjustments include the selection of an alternate program, the modification of program parameters, or the recalibration of a sensor. Control commands may be passed from higher-level logical sensors or from the Integration Controller. Each command is packaged as a control object, which has the following format:

1. **Destination logical sensor name:** Identifies the ELS for which the command is intended. If a command is intended for all subordinate logical sensors, then the destination name is **ALL**.
2. **Control command:** This is the actual command to be executed. It is expressed as an enumeration of a keyword string which is interpreted by the I/O Controller. The command may be one of a set of generic, system-wide commands, or may be specifically defined to work only with a particular logical sensor.
3. **Associated parameters:** A place is provided within the control object for parameters associated with each command.

4.2.2.5 Control Output

Control output from an ELS consists of control commands to lower-level logical sensors. These may be generated by the issuing sensor, or may be passed through from an ELS at a higher level.

4.2.3 Controller

The controller is comprised of three components which work together to supervise the internal operation of the ELS. These components, the Logical Sensor Controller, the Local Exception Handling mechanism, and the Local Knowledge Base are detailed in the following sections.

4.2.3.1 Logical Sensor Controller

The internal operation of the logical sensor is supervised by the LS Controller. The controller serves two main purposes: response to external commands, and internal monitoring and optimization of logical sensor performance through error detection and recovery. It is an extension of the Selector of the original logical sensor specification [20], which increases the functionality and robustness of the ELS through the use of a local knowledge base and exception handling mechanism. By internalizing specific operational knowledge, the ELS encapsulates the sensor operation.

The LS Controller provides the logical sensor with a mechanism to respond to commands passed from the I/O Controller. A number of standard control commands are defined for all logical sensors, as listed in Table 4.2. These, in addition to user commands, are stored locally for each ELS. A copy of user commands is also maintained by the Integration Controller. This provides controlling sensors with information about the capabilities of subordinate sensors.

Table 4.2: Standard logical sensor control commands.

Command	Description
INITIALIZE	Initializes the logical sensor upon creation.
CALIBRATE	Calls a predefined calibration routine for the logical sensor.
POLL	Provides a response to queries about the logical sensor properties. Returns the information stored as the logical sensor characteristics.
SENSE	Provides output in the form of the characteristic output vector. This output is dependent on both the state of the sensor inputs and the currently selected program.
RESET	Causes all of the logical sensor parameters to be reset to their initial values.
TEST	Calls one or more of the predefined embedded tests contained within the logical sensor.
SELECT	Causes an alternate program within the logical sensor to be selected, should one be available. The program is chosen by the Logical Sensor Controller – a specific program cannot be requested.
MONITOR	Validates the data contained within the Characteristic Output Vector through comparison with a predefined criterion.
USER	Allows user to send commands which are specific to a particular sensor or group of sensors.

Local knowledge of the operating characteristics of the ELS is used for program parameter

adjustment. For example, a request such as `INCREASE EDGES` to an edge detection ELS may be mapped to an appropriate change in mask size or adjustment of thresholds. This contrasts to a request such as `set mask_size = 3` which requires that the requesting program have knowledge of the specific algorithm in use and the effect of parameter changes.

The performance of the ELS is affected by the selected program and the adjustment of the program parameters. An alternate program may be selected in response to a sensor failure or in response to a command passed from a controlling sensor. In the case of a sensor failure, the alternate program selected typically relies on an alternate set of logical sensors for input. This redundancy provides a measure of robustness to the sensor system.

4.2.3.2 Local Exception Handling

The Local Exception Handling module is responsible for internal diagnostics, local error detection, and recovery. The testing and recovery schemes are limited to the domain of the ELS, using the methodology outlined in Section 4.3.4 with a relatively small set of tests and recovery schemes. Errors which cannot be handled locally result in the sensor issuing an error message.

The standard error messages are listed in Table 4.3. Typically, these errors are passed to the Integration Controller, which attempts to rectify the problem from a global, rather than local, perspective.

Table 4.3: Standard logical sensor error conditions.

Error	Description
TIME OUT	Unable to complete operation in allotted time.
OUT OF RANGE	Computed value outside of specified range.
OUT OF MEMORY	Operation requires more memory than is available from the system.
HARDWARE FAULT	Problem with hardware device.
NOTHING FOUND	Insufficient data to compute desired result.
GENERAL FAILURE	Category for all errors not explicitly defined.
USER DEFINED	Allows user to expand standard error types for a particular sensor.

4.2.3.3 Local Knowledge Base

The Knowledge Base is constructed as a logical sensor is created. Contained within each logical sensor, it contains a variety of information which is essential to the operation of the logical sensor. Among the information contained in the Knowledge Base are default parameters used during initialization and reset; command definitions, both local and standard; criteria for monitoring sensor performance; tests to determine error causes; local error definitions for sensor specific problems; and error mappings which are used to assist in error recovery. In general, this information is not available to other sensors or modules in the system.

4.2.4 Programs

Each ELS must contain at least one program to process the input data; however, when possible, each logical sensor may contain a number of alternate programs. There are two main reasons that multiple programs may be desirable within a logical sensor:

1. Multiple programs enable the use of different input sources and combinations thereof.
2. Different algorithms may be used to process the input data at different rates or with different degrees of precision. This provides a mechanism for *sensor granularity*. For example, a high-speed, coarse interpretation may be used in place of a low-speed, high-resolution interpretation in time-critical situations.

While the method of data generation may be different for each program within the ELS, each must be capable of providing data in the format specified by the COV. Programs may be either device drivers or processing algorithms, depending on the type of input handled. These are described in Sections 4.2.4.1 and 4.2.4.2 that follow.

4.2.4.1 Device Drivers

In the context of the ELS, device drivers are used only for direct interaction with physical sensors. The raw output signals from transducers are usually not in a form that may be used directly by a computer system. A device driver is used to interpret the raw signals from physical sensory devices. Output from digital transducers is obtained directly through a digital input device such as a data

acquisition board or frame grabber. Signals from analog transducers must first be digitized using an analog to digital converter.

Each physical device has an associated driver which, in addition to signal interpretation, manages the actual data transfer and control operations. This may include starting and completing I/O operations, handling interrupts, and performing any error processing required by the device. Further information on device drivers is provided by Baker [66].

IEEE P1451 compliant devices are treated in a similar manner. The major difference is that the driver is onboard the transducer. By interfacing using the Smart Transducer Object Model, the signal-level details are hidden. An ELS designed to work with a smart transducer will not require any modification if the transducer is exchanged for another designed for the same purpose.

4.2.4.2 Processing Algorithms

Processing algorithms are used to encapsulate signal processing routines. The encapsulation of signal processing routines is at the core of the logical sensor model. 'Virtual' devices may be constructed for sensors as diverse as line detectors, 'red' finders, and weight estimators by combining different sets of lower-level logical sensors in order to perform the task at hand.

Should sensor fusion be desirable for a particular application, it is performed by an ELS that is selected or designed for this task. Any fusion mechanism may be employed, though the discordance-based sensor fusion method presented by Murphy [9] is used herein for its robustness. For example, images of an object provided by multiple cameras positioned at different viewpoints may be fused and integrated in different ways. One algorithm may fuse images from the 'compass points' around an object to produce a continuous 360° view of the object. Another may integrate this fused image with an overhead view from another camera to validate the information from both sources in addition to detecting features that may otherwise be imperceivable. The use of such algorithms is considered by the first example in Chapter 6.

4.3 Integration

Integration involves the packaging of the sensory information provided by the logical sensors into a form suitable for the Inference Engine. Extracted information and features from top-level logical

sensors are used to provide high-level representations of the objects of interest. As this is the final stage before decisions are made based on the sensor data, particular attention is paid to ensure data integrity.

The specification and components for integration are given herein. However, the focus of this work is on the design of the object model, ELS, and Inference Engine. The implementation of the other components is left for future work.

4.3.1 Integration Controller

All top-level logical sensor outputs pass through the Integration Controller before entering the Inference Engine. The Integration Controller oversees the operation of the system, acting as an interface between the sensors and the Inference Engine. Here, the concept of what the system is trying to accomplish is maintained. It serves to coordinate sensor integration, in addition to data validation and exception handling activities which cannot be handled at the logical sensor level.

Sensor uncertainty is used throughout the integration process. Confidence measures are used for the identification of sensing errors and for the integration of sensor data. Sensor performance criteria are maintained in the system Knowledge Base. These criteria are used to determine whether the data provided by the sensors lies within acceptable ranges or is of an expected form. All data which is successfully validated is passed to the Inference Engine; problematic data is passed to the Diagnostics module.

As problems are encountered at the ELS level, this information is passed to the Integration Controller. The controller uses the Diagnostics module and information contained in the Knowledge Base to determine the appropriate corrective action. This may involve sending out commands to adjust logical sensor parameters, recalibrate logical sensors, or reconfigure the sensor hierarchy. The removal of malfunctioning sensors from the hierarchy or a reordering of sensors are among reconfiguration possibilities.

4.3.2 Validation

The Validation module is used to perform high-level verification and validation of the sensor information provided by the logical sensors. While this may be as simple as determining if the sensor

data lies within acceptable ranges or is of an expected form, such tests are usually performed at the logical sensor level. Instead, the Validation module attempts to detect disparities between the information being provided by multiple sensors.

Most systems tend to use a small set of sensors. There may be some redundant sensing capability; however, the majority of sensors are likely to be complementary. This makes the validation of information difficult because there may not always be an alternative sensor that can corroborate a suspect sensor. This is handled by making inferences from the behaviours of other sensors. Sensor performance criteria and other expert knowledge for sensor validation is maintained in the system Knowledge Base.

If an error or disparity is detected, the problem is passed to the Diagnostics module which then attempts to determine the cause of the failure and provide a solution. All data which is successfully validated is passed to the Inference Engine.

4.3.3 Diagnostics

Should a problem be identified during data validation or an exception cannot be resolved at the logical sensor level, the Diagnostics module coordinates with the Exception Handling Mechanism to determine the exact nature of the problem and implement possible solutions.

The Diagnostics module may be viewed as an exception controller. It interfaces with the Integration Controller and Validation modules which identify error conditions and the Exception Handling Mechanism which contains information for error classification and recovery.

When a sensor fails, the Diagnostics module queries the Exception Handling Mechanism for a list of possible hypotheses which may explain the cause of the sensor failure. It then carries out the specified tests until a particular hypothesis can be confirmed.

Upon determining the cause of the error, the Exception Handling Mechanism provides a recovery method. This method is then implemented by the Diagnostics module to rectify the problem.

4.3.4 Exception Handling

Exception handling provides support for the Diagnostics module which aims to maintain the successful operation of the system in the event of sensor failure. Exception handling routines are

invoked when data fails to satisfy a predetermined constraint or is in conflict with data from another sensor. Sensing failures must be handled expeditiously to allow the system to continue to operate effectively. In automated inspection applications, it is generally unacceptable for products to pass by unevaluated or to slow/stop the line in order to resolve sensing failures.

As stated above, exceptions are handled by first classifying the nature of the error, as discussed in Section 4.3.4.1. Once classified, an attempt is made to rectify the cause of the error using the recovery scheme outlined in Section 4.3.4.2.

It is worth noting that the system does not assume that any sensors used for error classification and recovery are themselves operational. Before each is used it must be functionally validated in advance.

4.3.4.1 Error Classification

Without the availability of a complete causal model, detected errors must be classified so that the appropriate corrective action may be taken. To simplify classification, it is assumed that there is only one sensing failure at a time. Sensor failures are classified into three types as follows:

1. Sensor malfunctions: This occurs when one or more sensors are malfunctioning. Examples include power failure, impact damage, miscalibration, etc.
2. Environmental change: One or more sensors are not performing properly because the environmental conditions have changed since sensor configuration and calibration. This often leads to precision errors.
3. Errant expectation: Sensor performance is poor because the sought object is occluded or lies outside of the sensor's 'field of view.'

Error classification is accomplished by a generate and test algorithm [67, 68]. The suspect sensors are first identified. An ordered list of possible hypotheses explaining the sensor failure is then generated. Each hypothesis is associated with a test which may be used for verification. These tests are performed in an effort to confirm or deny the proposed hypotheses. This process is repeated until a hypothesis is confirmed.

The generate and test method does not require formal operators for the generation of hypotheses. This allows the system to use a rule-based method to select from a list of candidate hypotheses. Unfortunately, this method can be time consuming if there is a large problem space and all hypotheses must be generated. This disadvantage may be overcome by constraining the problem space, thereby limiting the number of hypotheses and reducing processing time. Testing is conducted until all tests have been performed or an environmental change has been detected. When the classifier is unable to resolve the cause of the error, the cause is assumed to be an errant expectation.

4.3.4.2 Error Recovery

For each error cause, there would ideally be a number of different recovery schemes. From these, the most appropriate would be selected by the exception handling mechanism. To limit the scope of the problem and reduce the overall recovery time, a direct one-to-one mapping of error causes to recovery schemes is utilized. A library of cases allows for the instant mapping of error cause to recovery scheme based on the error classification.

Functions are used to repair individual sensors or reconfigure the sensor hierarchy. The sensor parameters are adjusted first — recalibration is accomplished by invoking a predefined sensor calibration routine. If the sensing configuration cannot be repaired through parameter adjustment or recalibration, the sensor hierarchy is altered. The alteration may suppress a particular sensor or remove sensors from the hierarchy.

4.4 Inference Engine

Once the sensory information collected by the logical sensors has been validated, it is passed to the Inference Engine. Here, based upon the examination of the extracted objects and features, decisions are made regarding the actions to be taken with each object.

The sensor inputs are used to form the antecedents of the control decisions to be made in the Inference Engine. The consequents of these rules are the actual decisions. These are passed from the Inference Engine to the Post Processor for conversion into action.

As shown in Figure 4.3, the Inference Engine divides the inference task into two parts. First, the information available from the various sensing devices is fed to the Inference Engine as the primary

input. This sensor information is used by the first module to determine a measure of certainty that the object is of each classification. These classifications with corresponding certainties are then passed to the second module.

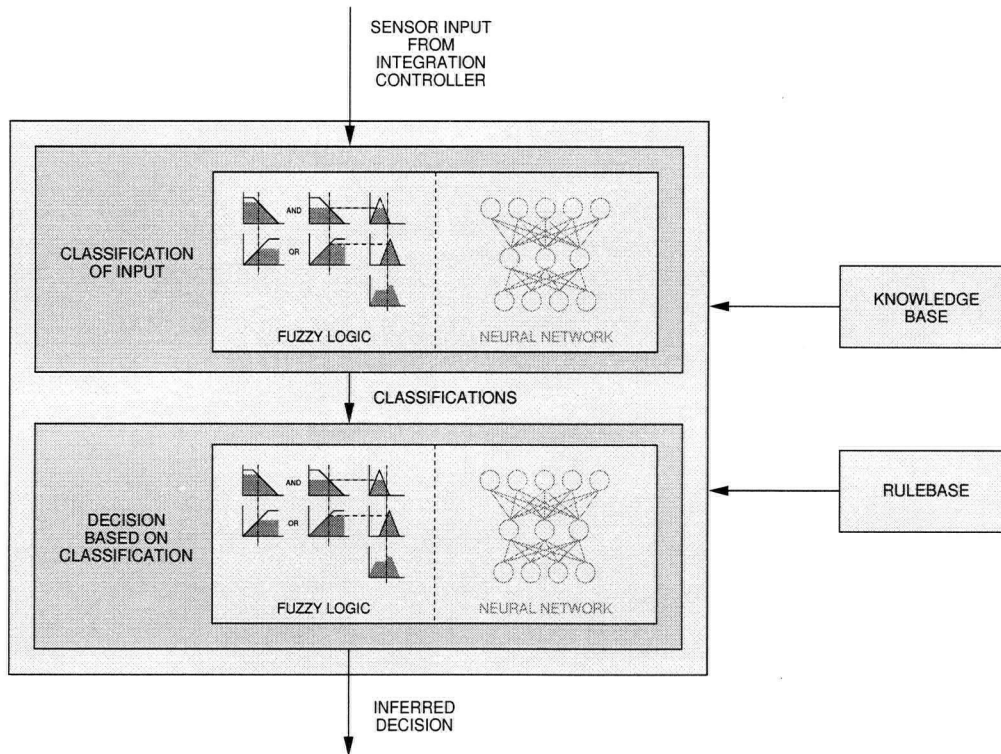


Figure 4.3: The Inference Engine used by ELSA. Inferences using fuzzy logic draw upon information contained in the Rulebase. The neural network-based inference mechanism (shown inactive) utilizes weights stored in the Knowledge Base.

The second module uses these classifications to infer a decision. If an object classification is certain, the decision is unambiguous. The advantage of this approach is evident when dealing with borderline cases. By considering the certainty measure for each object classification, an appropriate decision may be made under uncertain conditions.

In this work, the Inference Engine is cognitive-based, using fuzzy logic [64] to make decisions. The advantage of this approach is that it allows the incorporation of expert domain knowledge. This expert knowledge may be formulated into a rulebase to serve as the basis for fuzzy inference. The base class which serves as a template for the development of the Inference Engine is outlined in Appendix D.

While fuzzy logic is the inference method currently used, other knowledge based systems could be employed. For applications where expert knowledge is less concrete, a feature-based inference technique such as artificial neural networks [69–72] could be used to interpret the sensor information and produce control decisions. Applications for neural networks include the analysis of infrared spectral data to determine the composition and moisture content of a product, and the chemical analysis of samples to determine quality or taste [5]. For these applications, the network must be interactively trained to produce the desired results. Other possibilities for feature-based inference techniques include Bayesian reasoning and the Dempster-Shafer theory of evidence.

4.4.1 Rule/Knowledge Base

Fuzzy logic and knowledge based inference rely upon expert domain knowledge supplied by the user. For grading and inspection tasks in particular, the expert knowledge available from human inspectors is available to the system designers. The Rulebase stores this repository of domain knowledge in the form of antecedent/consequent rules. For example, a fruit classification system may include the following simple rulebase:

IF Shape IS *round* AND Colour is *red* THEN Fruit = *apple*

IF Shape IS *round* AND Colour is *orange* THEN Fruit = *orange*

IF Shape IS *elongated* AND Colour is *yellow* THEN Fruit = *banana*

In the case of fuzzy logic, linguistic variables, such as *round* and *red* are associated with membership functions that describe a fuzzy subset of the universe of discourse. These fuzzy sets are also stored in the Rulebase. Each set defines the universe of discourse and membership functions for each subset that corresponds to a linguistic variable. Membership functions may be triangular, trapezoidal, Gaussian, etc.

The Knowledge Base contains a diverse set of information that is used by the Integration Controller and, depending on the inference mechanism, the Inference Engine. In the case where a neural networks Inference Engine would be implemented, the network topology and the trained weights between the hidden layer(s) and output layer would be stored here. Other information contained in

the Knowledge Base consists of the object model, control commands, error conditions, ELS characteristics, and sensor performance criteria. This information is used by the Integration Controller to oversee the operation of the logical sensors. Performance criteria are used to validate sensor data and reconfigure the hierarchy in the event of a sensor malfunction.

4.5 Post Processing

Once the inference engine has processed the sensory information and interpreted it, any decisions made must be converted into actions. This involves the conversion of a directive into a plan of action for execution. For example, the decision to place a bruised apple into the 'bruised apple bin' must be translated such that the appropriate actuators affect this action at the appropriate time. The Post Processor acts as an interface between the Inference Engine and the drivers which are used to control the process machinery. Drivers are then used to convert control actions from the Post Processor into the specific format required by each device. The possibilities for devices which may act as process machinery are countless. Devices may range from simple actuators such as solenoids and electromagnets, to complex systems such as multiple degree of freedom robotic manipulators. However, the issues involved with post processing are beyond the scope of this work and will not be addressed further.

4.6 Summary

In this chapter, the organization of the Extended Logical Sensor Architecture (ELSA) was presented. Each component was introduced and its role within the architecture was described. Together these components comprise a modular, scalable, and robust system. Sensory information is encapsulated by Extended Logical Sensors. The integrity of the sensor data is ensured by the Integration Controller working in concert with the Validation and Diagnostics modules. Process decisions are made by the Inference Engine on the basis of the validated sensor information. The following chapter will discuss the construction of a system based on ELSA.

Chapter 5

Construction Methodology

To maximize system robustness and usability, the construction of an industrial sensing and processing system using ELSA follows a set procedure. An overview of this methodology is presented in Figure 5.1. The sections that follow detail the various phases of the process. The methodology will be further illustrated by the example applications provided in Chapter 6.

5.1 Problem Definition/Requirements Specification

The first phase of the design process involves the recognition of the needs of the particular industry or process. These needs often arise from dissatisfaction with the existing situation. They may be to reduce costs, increase reliability or performance, or to adapt to customer expectations.

From the needs, a clear statement of the problem to be solved may be formulated. This problem definition is more specific than the general needs; it must include all of the specifications for what is to be designed. Hence, the designer must consider what the capabilities of the system should be. Following the general principles for system design outlined in [73], a set of minimum functional requirements is specified. By definition, these requirements should focus on the *functions* of the design without overspecifying property values and performance parameters. This ensures that the design process is not forced to follow a predetermined path.

Often the requirements of the system may be considered in four categories [74]:

1. *Musts*: Requirements which must be met.

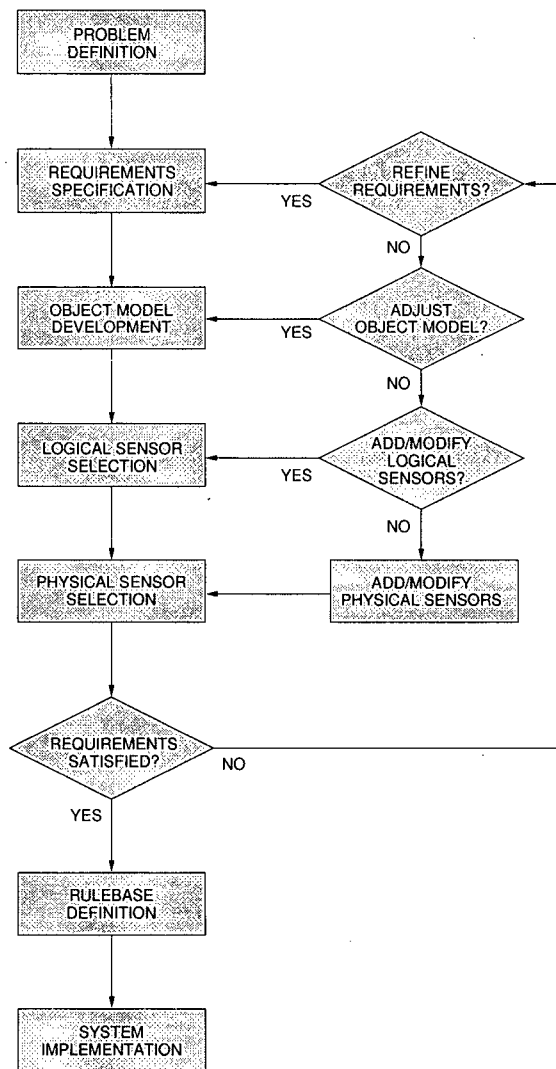


Figure 5.1: Overview of construction methodology.

2. *Must nots*: Constraints on what the system must not do.
3. *Wants*: Requirements that are desirable but not essential.
4. *Don't wants*: Specifies what, ideally, the system will not do.

These requirements would typically include performance (speed, accuracy, etc.), cost, maintainability, size, weight, complexity, standards and regulatory requirements, customer preferences, and market constraints, among others. The articulation of these requirements is used as a guide for subsequent phases. If any of the requirements are left unsatisfied, the design is inadequate. The requirements also serve to keep the design focused on what is necessary for the task at hand.

5.2 Object Model Development

Object model development for ELSA is a two-stage process. First, based upon the requirements of the system from the previous phase, the primary features or characteristics upon which classifications are to be made are identified. As discussed in Chapter 3, it is advantageous to keep the size of this set to a minimum. Typically, the features in this set are at a high level of abstraction. They occupy the top of the feature layer of the model (right side of Figure 3.1). From this set, each feature which is not atomic is decomposed into a set of subfeatures. This decomposition continues until all features are atomic. A feature is considered to be atomic if it cannot be subdivided further. This process is illustrated in the upper-half of the flowchart in Figure 5.2.

Once high-level features are represented by atomic features in the lower section of the object model, the high-level information is used to define the object classifications following the steps in the lower-half of Figure 5.2. The classifications occupy the upper level of the model topology (left side of Figure 3.1). Each object classification is defined by first specifying the relevant primary features with fuzzy links. The fuzzy links to each classification are then associated with a fuzzy descriptor. These descriptors specify to what degree of confidence the particular primary features must be identified to be confident in the object classification. The complete algorithm used to construct an object model is as follows:

1. Select an object to model.
2. Determine the primary features of the object.
3. Select a primary feature.
4. If feature is atomic, goto 9.
5. Determine subfeatures.
6. Select a subfeature.
7. If feature is not atomic, goto 5.
8. If there are additional subfeatures, goto 6.

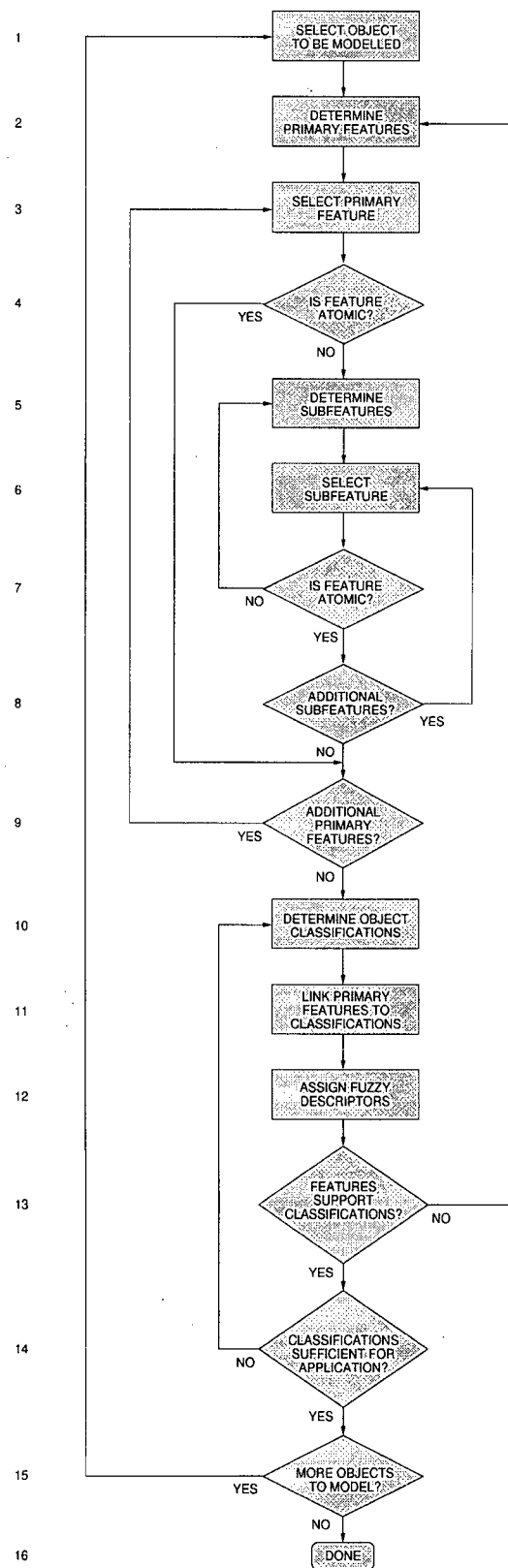


Figure 5.2: Object model development methodology.

9. If there are additional primary features, goto 3.
10. Determine desired classifications of object.
11. Link primary features to object classifications with fuzzy links.
12. Associate fuzzy descriptors with each fuzzy link.
13. If the defined primary features do not support the object classifications, goto 2.
14. If the defined object classifications are not sufficient for the application, goto 10.
15. If there are additional objects to model, goto 1.
16. Done.

The classification layer of the object model (relevant features in combination with relative weights) serves as a template for the Inference Engine which, in practice, makes the classification decisions based on the feature information extracted by the logical sensors. The development of the Rulebase is described in Section 5.4.

5.3 Logical/Physical Sensor Selection

The selection of logical sensors is driven by the primary, intermediate, and atomic features that have been identified as necessary for the object model. Sensor selection starts with the primary features. Each feature has a corresponding ELS which packages the information from lower-level sensors (logical or physical) into the representations used for object classification. Many of the low-level logical sensors are selected from a reusable ELS library. The logical sensors contained within the library perform standard image and signal processing operations. The algorithm for constructing the ELS hierarchy, Figure 5.3, is as follows:

1. Select a primary feature from the object model.
2. Define a LS to provide primary feature.
3. If feature is atomic, goto 7; else, continue.

4. Select a subfeature.
5. Select or define a LS to extract feature.
6. If feature is atomic, goto 7; else, goto 4.
7. Does LS receive input directly from a physical sensor? If so, goto 9; else, continue.
8. Select or define logical sensors required to supply information to LS that provides atomic feature. Goto 7.
9. Select required physical sensor.
10. If there are additional subfeatures, goto 4.
11. If there are additional primary features, goto 1.
12. Done.

Physical sensors are selected to satisfy the input requirements of the LS associated with each atomic feature. This requires a consideration of both the input requirements and the capabilities of available transducers. A feature that is beyond the range or capabilities of a single sensor may be accommodated by the fusion of data from multiple sensors which cover the feature space. A LS is then defined which provides the feature, fusing the data from each of the physical sensor inputs.

Other considerations include whether the system should attempt to utilize a single sensor for multiple tasks or whether specialized sensors will be used. For example, a camera can provide size, colour, and shape information. Clearly, separate cameras are not required to extract each of these features. Using visual information and a correlation between length, area, and mass, a weight LS may be defined to estimate the weight of an object. Depending on the application, this may be used to replace or augment the information from a load cell.

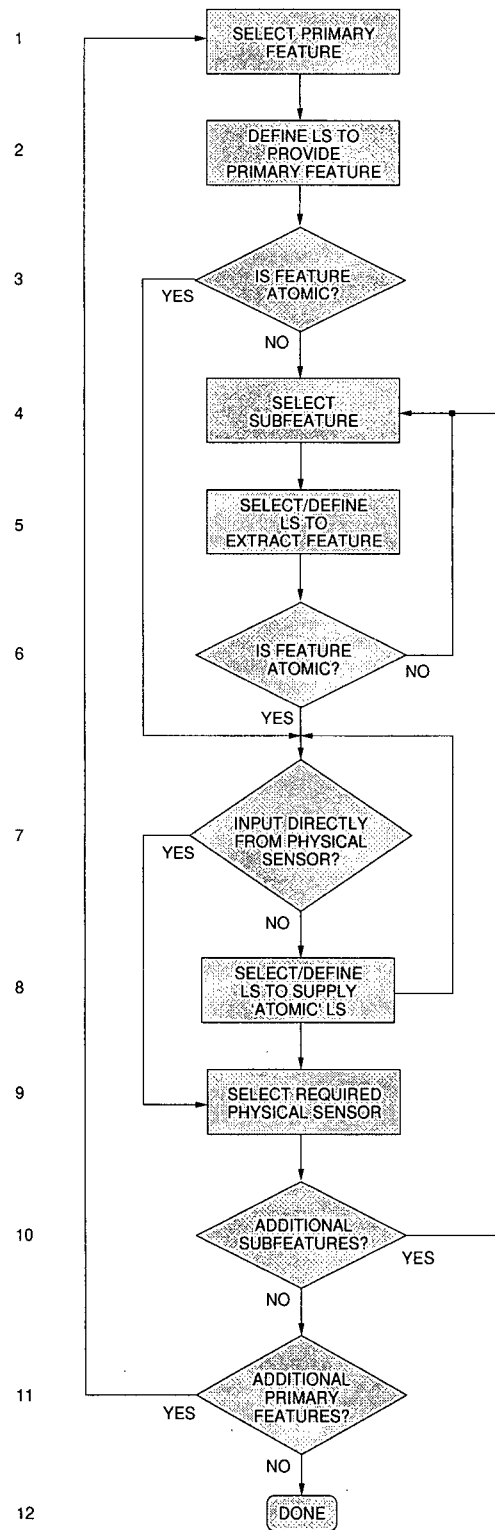


Figure 5.3: Methodology for the development of the ELS hierarchy.

5.4 Rulebase Definition

The Rulebase defines both rules for object classification and rules to infer the appropriate system output from these classifications. It is generated directly from the object classifications contained in the object model.

The classification rules use the fuzzy descriptions of each classification as the basis for description. The confidence in the detection of each primary feature may then be used as input to the classification rules. Each rule expresses a degree of confidence in the classification of the object based on the detection of the primary features. The rules for each classification are combined using the compositional rule of inference, e.g. using a *sup-min* operation [63], to produce a measure of confidence that the object is of each classification.

Conversion of the representation in the classification layer of the object model into a rulebase which may be used by the Inference Engine is accomplished using the following algorithm, Figure 5.4:

1. Select an object classification.
2. Use fuzzy links to identify the primary features that this classification depends on.
3. Determine the interdependencies of primary features. Each rule is defined using the minimum number of features. For example, consider a classification which is dependent on three primary features. If one of these will result in object being classified as belonging to the given classification, regardless of the other two, rules are defined that contain only this feature. Other rules will contain both of the other features, provided that the presence of each is required for proper classification. Primary features may be combined with *AND* and *OR* operators.
4. Specify rules which correspond to the fuzzy descriptors used to describe the object classification. These describe conditions necessary for a high confidence in the detection of the particular classification. These are mandatory.
5. Specify rules which are opposite to the fuzzy descriptors used to describe the object classification. These describe conditions which indicate that the classification is not applicable to the object. These are mandatory except for the case of a default classification — in other words,

- a classification for those objects that do not satisfy the criteria of the other, more specific, classifications.
6. If classifications with lower confidence should be considered to increase the robustness of the system, continue; else, goto 8.
 7. Specify rules having fuzzy descriptors which correspond to a low degree of confidence in the detection of one or more primary features.
 8. If there are additional classifications, goto 1.
 9. Done.

Decision rules are defined to inform the system what the should be done according to how each object is classified. Decisions are defined using the confidence in each object classification as the antecedent(s); the appropriate decision(s) forms the consequent. For industrial systems, the decision often corresponds to an action to be taken. A grading system may decide to place objects into particular bins, based on how they are classified. If an object classification is certain, the appropriate decision is straightforward. By evaluating the confidence of each object classification, borderline cases may be handled in the most appropriate manner.

The decision rules are defined in a manner similar to the classification rules, though they are based on the object classifications rather than the primary features. Figure 5.5 illustrates the algorithm that follows:

1. Determine decisions which may be made based on object classifications. Ensure that there is a decision that corresponds to each classification.
2. Select a decision.
3. Identify the classifications upon which decision depends.
4. Specify rules for each classification that, when identified with a high degree of confidence, result in the decision.
5. If classifications with lower confidence should be considered to increase the robustness of the system, continue; else, goto 7.

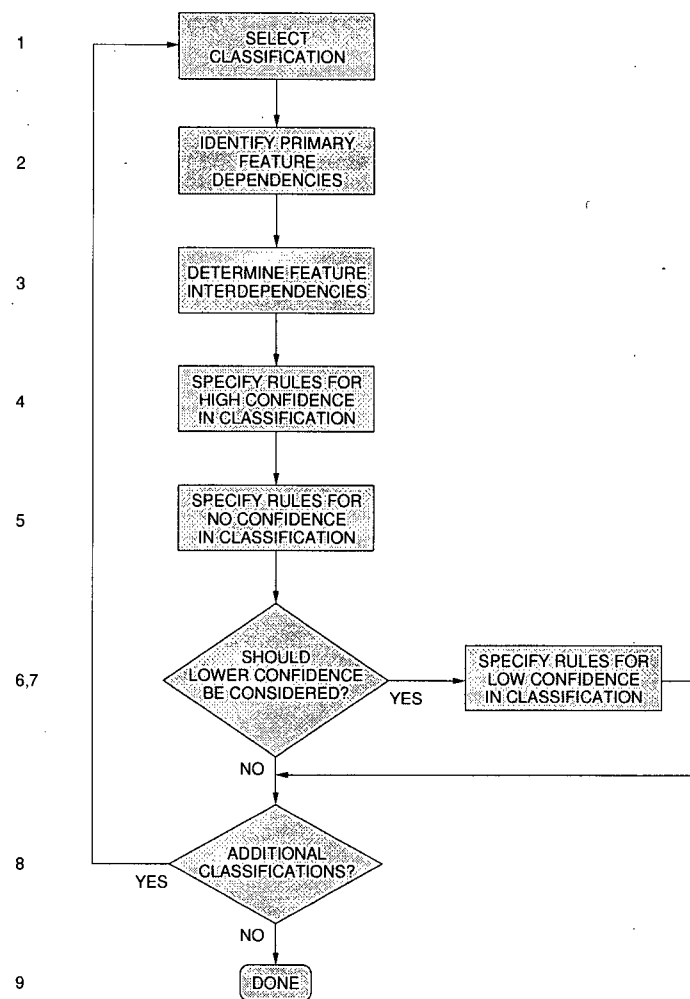


Figure 5.4: Methodology for the definition of the rulebase for object classification using the object model.

6. Specify rules that define a decision based on a classification or classifications that have been identified with a low degree of confidence. This may be used to eliminate false positives by rejecting borderline cases. Depending on the application, low confidence in a single classification may be sufficiently serious; for others, an ambiguity (low confidence) in two or more classifications may be required.
7. If there are additional decisions, goto 2.
8. Done.

Inferring a decision from the object classifications uses a methodology similar to that used for

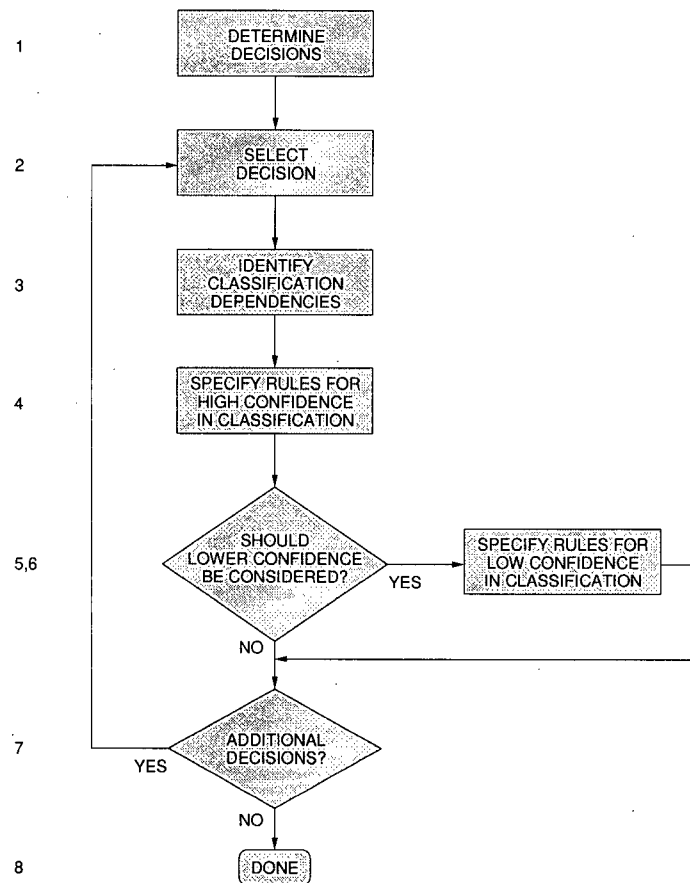


Figure 5.5: Methodology for the definition of the decision rulebase based on object classifications.

determining the confidence in the detection of primary features, as discussed in Section 3.6.3.2. As shown in Figure 5.6, membership functions *no*, *low*, and *high* specify the degree of confidence in the classification of an object.

5.5 System Implementation

Having completed the functional requirements analysis, defined the object model, chosen the logical sensors and physical sensors, and defined the rulebase, the next stage is to realize and integrate these components to produce a working system. The following steps indicate the various stages in this process:

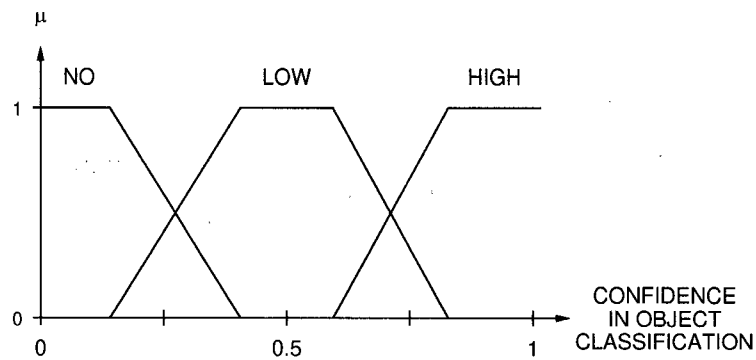


Figure 5.6: Membership function used to represent confidence that an object is of a particular classification.

1. Construct the physical system. This includes the arrangement of physical sensors as well as product delivery and handling systems.
2. Select the required ELSs that are available from the library.
3. For ELSs that are required but are unavailable from the library, these must be constructed. The ELS base class, used as a template for ELS construction, is presented in Appendix B.
4. Implement the rulebase and associated membership functions using the classes described in Appendix D.
5. Implement the object model using the object class described in Appendix A. This is stored in the Knowledge Base.
6. Define the Validation module providing parameters by which the sensor information may be evaluated.
7. Define the Exception Handling Module, providing tests used for error classification and error recovery schemes (mappings).
8. Implement the Integration Controller to coordinate sensor integration and drive the system operation.
9. Select the inference mechanism(s) used by the Inference Engine. Define these if necessary.
10. Implement post processing and control as required by the application.

As is apparent, further work needs to be done towards the automation of these steps. This would improve the ease with which a system may be constructed using the ELSA methodology. While the system construction is not currently automated, each component has been designed with this goal in mind. Future automation efforts should not require any significant redesign of the various modules and components that comprise ELSA.

5.6 Modification and Refinement

Once the system has been constructed, it may be necessary to modify or refine some of the components. Typical changes include the following:

- Rulebase alteration.
- Membership function tuning.
- Addition or change of classification.
- Addition or change of primary features.
- Addition, change, or removal of physical/logical sensors.

One or all of these may be necessitated to improve the performance of the system, to account for deficiencies in the original design, to adapt to changing specifications or customer requirements, to incorporate different or new sensor technologies, to modify the system for a different application, or some other unforeseen need. The hierarchical structure of the object model and sensors ensures that changes remain local — the structure as a whole is unaffected.

The simplest changes involve the adjustment, addition, or removal of rules from the rulebase. These changes are made to fine-tune the system or to infer different decisions from the sensor information. These changes do not affect any other part of the system. New rules may require additional membership functions to be defined.

If it is found that the granularity of a membership function is insufficient, or shape (range, mean, function) does not properly reflect the linguistic variable(s), the membership functions may be tuned. Tuning will affect all rules which use the membership function. If the changes are

substantial, such as the addition or removal of linguistic variables to modify the granularity, each dependent rule may have to be reevaluated. Rules that do not make use of the membership function are unaffected.

The object model may be adjusted by adding new object classifications. An additional classification will not affect any others; it is simply linked to the appropriate primary features. Additional rules will have to be defined for the new classification. Modification of existing classifications may be achieved by creating links to unused primary features or by adjusting the fuzzy descriptors. Each will require the rules that correspond to the classification to be updated. Such modifications may be necessary if objects are being improperly classified.

If after tuning, or adding new classifications, objects are still improperly classified, it may be necessary to define an additional primary feature. Additional features should be chosen such that objects can be differentiated on the basis of characteristic features. The definition of a new primary feature will follow the same procedure outlined in Section 5.2. New subfeatures and physical sensors may be required. The existing sensor hierarchy is not affected.

Problems with feature extraction are handled through the adjustment of the ELS(s) associated with the feature. Adjustments may include refinement of properties and relations or alteration of parameters. Should these prove unsuccessful, the ELS may be replaced by another providing the same function or the sensor hierarchy may be redefined. Such a redefinition would only affect those sensors associated with the feature. If it is low-level feature, higher-level features are oblivious to any changes.

Finally, a new physical sensor may be added to the system. This could be to replace an existing sensor or to augment the system capabilities. Sensor replacement will only require a new ELS to encapsulate the sensor. An additional sensor will require, at minimum, a new ELS but may require the sensor hierarchy, object model, and rulebase to be redefined to take advantage of the new information.

5.7 Summary

This chapter has outlined the basic steps in the design of an ELSA-based multisensor integration system for a particular industrial application. These steps include:

1. Identification of the problem.
2. Specification of the functional requirements.
3. Development of the object model.
4. Selection of appropriate Extended Logical Sensors and physical sensors.
5. Definition of the classification and action rules from which to infer process decisions.
6. Implementation of the system.

Once the requirements of the system have been determined, the object model is defined to represent the features and classifications of the objects that the system must deal with. The selection of sensors and the specification of the rules used by the Inference Engine follow directly from the object model. This process serves to isolate the user from the technical details of the system design and construction. This process is further illustrated by the examples in the following chapter.

Chapter 6

Application Examples

This chapter provides examples of the construction of multisensor integration systems for industrial inspection. Two examples, drawn from industry, are considered. These examples are not attempts to create fully automated industrial working prototypes, but rather to illustrate how the ELSA methodology could be used to construct a sensor integration system for product inspection.

The first example, metal can inspection, is an illustrative example which deals with the inspection of a uniform object. The second example is herring roe grading. The non-uniform nature of this product introduces a number of interesting automation challenges. These examples are selected to contrast each other: the first example is simple to model but utilizes a relatively large number of sensors; the second model is more complex to develop but requires fewer sensors. For each, the object model, the ELS hierarchy, and the Inference Engine are developed using the ELSA approach.

6.1 Can Defect Detection

6.1.1 Background

A wide variety of food products are packaged in sealed rigid metal cans. The majority of cans are sealed using a machine called a double seamer. This machine interlocks the can lid and body forming a double seam. Seaming compound is used between the layers of interlocking metal to complete a hermetic seal. Most cans are sealed under a vacuum. The integrity of these cans may be compromised by a wide variety of defects. Improperly sealed cans can lead to botulism. Defects

may arise at any one of the stages of can manufacture; namely, filling, closing, processing, and handling, before the can reaches the customer.

Defects are classified as serious if there is visual evidence that there is microbial growth in the container or the hermetic seal of the container has been lost or seriously compromised [75]. There are a number of possible serious defect classifications. Most of these are related to the proper formation of the double seam. Examples include: seam inclusions, knocked-down flange (KDF), knocked-down end (KDE), knock-down curl (KDC), pleats, vees, puckers, side seam droop, cut-down flange, and dents. The majority of these are visible from a side view of the can, Figure 6.1; others from a top view, Figure 6.2.

6.1.2 Problem Definition/Requirements Specification

The current system for the automated inspection of metal cans uses equipment to measure the weight of each can and a double-dud detector which mechanically measures the amount of deflection of the can lid. The deflection is used as a measure of the amount of vacuum in the can. A well-sealed can will maintain a vacuum internally — the lid is deflected inwards (concave) by the vacuum. Improperly sealed cans exhibit less concavity. Cans which exhibit vacuum or weight values outside of statistically determined limits are ejected for manual inspection.

Unfortunately, a number of potentially serious defects may go undetected as vacuum may be lost at a later time during shipping, handling, or storage. To address this issue, it is proposed to augment the current configuration with a vision system capable of detecting many of the double seam defects.

Ideally, such a system would be used as part of a company's Hazard Assessment at Critical Control Point (HACCP) strategy. Cans passing through this system would have to pass each of the individual tests (weight and vacuum) already outlined and established through industry guidelines. This integrated system would then provide a secondary quality assurance check to identify those cans which slip through the individual tests.

The target application is the inspection of half-pound (227 g) salmon cans. These are typically two piece cans: a bottom and sides drawn from a single piece of metal with a separate stamped lid. The two are sealed together using a double seaming machine just after filling.

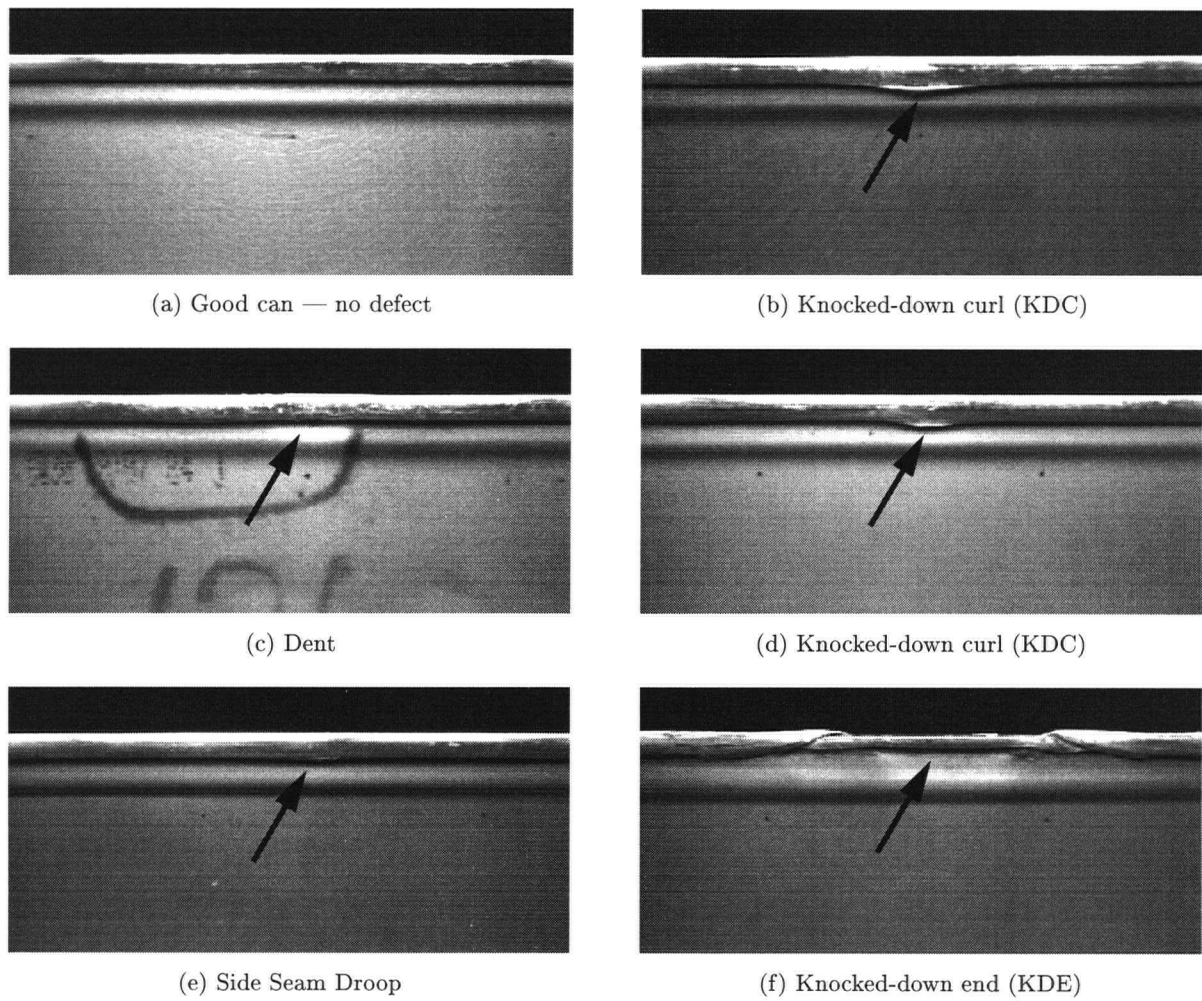


Figure 6.1: Examples of canner's double seam defects — side view.

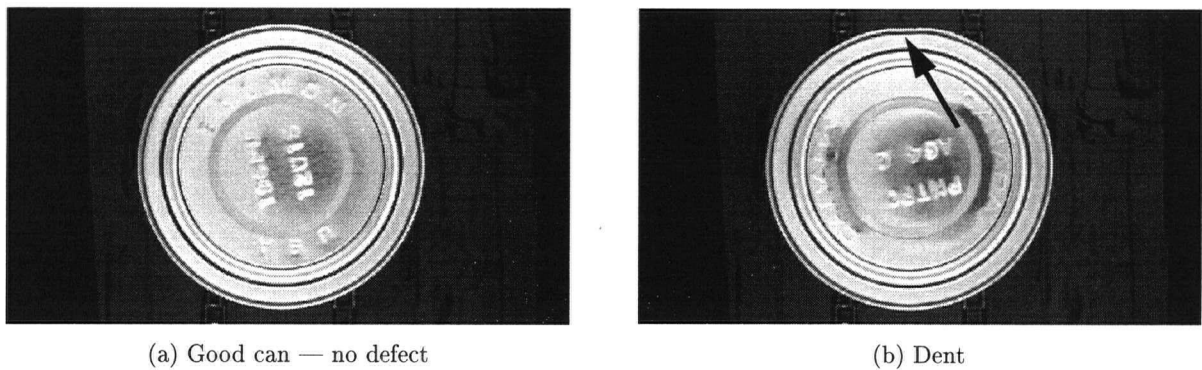


Figure 6.2: Examples of canner's double seam defects — top view.

The general sensing requirements of the multisensor system for the inspection of sealed metal salmon cans are as follows:

1. Detection of cans which exhibit insufficient vacuum (top lid deflection < 1 mm).
2. Detection of cans which are under weight (< 227 g).
3. Detection of cans which are over weight (> 235 g).
4. Detect double seam defects of top lid visible from either above and/or the sides of the can.
5. The occurrence of false positives should be minimized as much as possible. Cans ejected from the system would still be hand inspected. An overload of false positives would negate the benefits of the system.

For the purpose of this example, these shall be considered as the minimum functional requirements of the system. Other requirements, such as the speed, cost, and reliability of the system are also important; however, they will not be addressed directly.

6.1.3 Object Model Development

From the developed functional requirements, three primary features may be defined. These are weight, vacuum, and seam defects. Of these, weight is atomic and not dependent on other features. Vacuum cannot be measured directly (to do so would compromise the seal integrity) and a subordinate feature must be defined. The top lid deflection is used as an indirect measure of the amount of vacuum in the can.

Seam defects vary widely in manifestation; however, all are characterized by deviations in the expected profile of the seam. Deviations may occur over the entire seam length (too thick or too thin), or may be local. Thus, the features are simply deviations (defects) in the seam as viewed from the top of the can and from the side. As shown in Figure 6.3, the seam defects may be broken down into features visible from the top and those visible from the side. These may be further broken down into the atomic components which permit the detection of these defects.

The primary features are combined to produce four object classifications: good, improper seal, underweight, and overweight. The good classification depends on all of the primary features. It

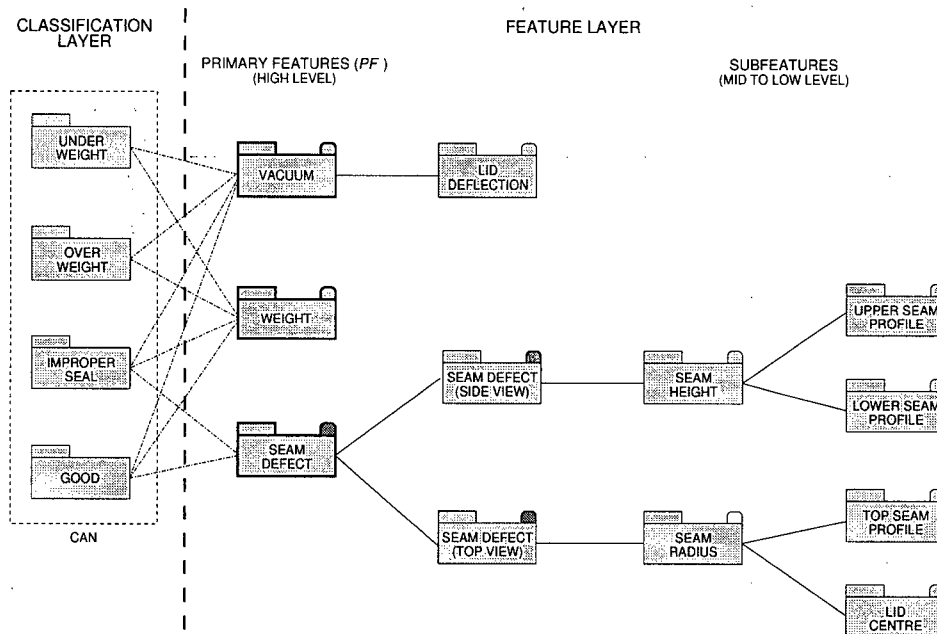


Figure 6.3: Object model for metal can inspection.

is defined as a can having average weight, average to high vacuum, and a low confidence in the presence of a seam defect. Similarly, an improper seal may be identified using a combination of the can weight, lid vacuum, and the detection of seam defects. This classification includes cans which exhibit seam defects as well as those cans that are normal to low in weight and have a low vacuum. Underweight cans have low weight and average to high vacuum; overweight cans have high weight and low to average vacuum. Vacuum is included in the underweight and overweight classifications as a measure of redundancy. An underfilled (and thus underweight) can exhibits a greater degree of vacuum; an overfilled may not allow the lid to deflect — affecting the vacuum measure.

6.1.4 Logical/Physical Sensor Selection

From the object model, a logical sensor hierarchy is constructed, Figure 6.4. The selection of sensors for the measurement of weight and vacuum is straightforward. A checkweigher automatic scale is used to measure the can weight. This is encapsulated by the weight ELS. Vacuum is determined indirectly by a double-dud detector. The lid deflection ELS, encapsulating the double-dud detector, passes the measured deflection to the vacuum ELS. This sensor then correlates the measured deflection to the amount of vacuum present.

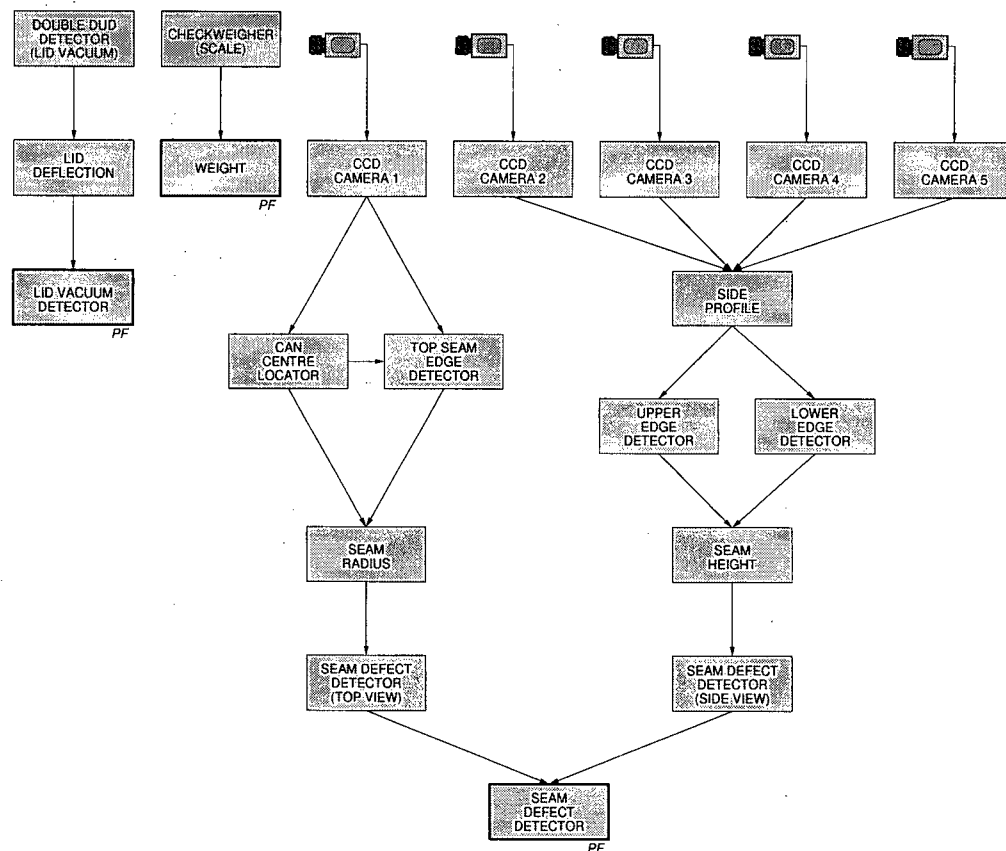


Figure 6.4: Logical sensor hierarchy for metal can inspection. Sensors which provide primary features are outlined in bold and tagged *PF*.

The seam defect ELS combines information from the side seam defect detector ELS and the lid seam defect detector ELS. This integration not only ensures that defects visible from only one viewpoint are detected, but apparently marginal defects which appear at the same location (around the circumference) in both views may be properly classified as serious. The logical sensors used to extract the lid and side seam profiles are based on image processing algorithms developed by Lee [76] for the purpose of metal can inspection.

Integration of complementary sensor information is performed by the side profile ELS to produce a view of the complete 360° circumference of the can. The results of this operation are shown in Figure 6.5. The seam defect detector ELS combines defect location information (expressed in polar coordinates about the can centre) from the lid seam defect detector LS and the side seam defect detector to better isolate borderline cases.

The logical sensors defined to extract seam defects require a total of five CCD cameras. A single

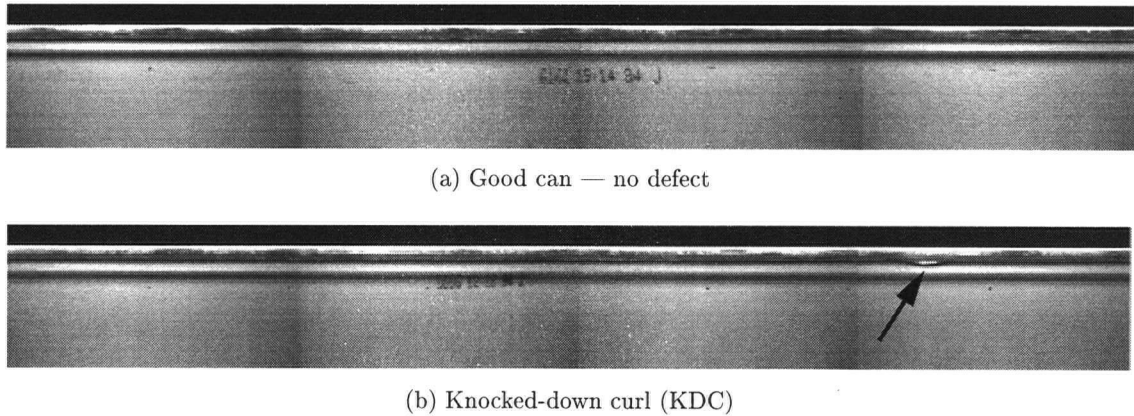


Figure 6.5: Full view of can sides reconstructed from four viewpoints.

camera is used to image the top view of the can, while four cameras are used to fully cover the circumference of the can when viewed from the side. JVC TK1070U colour CCD cameras were used. The top camera utilized a 12.5 mm f1:1.3 lens; the side cameras were equipped with 75 mm f1:1.8 lenses with a 5 mm extension.

6.1.5 Rulebase Definition

The rulebase generation follows from the object model. The object classifications outlined in Section 6.1.3 are used as the basis for the classification rules, Figure 6.6.

The decision rules, Figure 6.7, are defined by simply rejecting all cans which, based on their classification, are clearly defective or are borderline cases. The consequent is the fuzzy singleton *reject*. The fuzzy membership functions associated with these rules are shown in Figure 6.8.

IF Weight IS *very low* THEN UnderWeight = *high*
IF Vacuum IS *high* AND Weight IS *low* THEN UnderWeight = *high*
IF Weight IS *low* THEN UnderWeight = *low*
IF Weight IS *high* THEN UnderWeight = *no*

IF Weight IS *very high* THEN OverWeight = *high*
IF Vacuum IS *low* AND Weight IS *high* THEN OverWeight = *high*
IF Weight IS *high* THEN OverWeight = *low*
IF Weight IS *low* THEN OverWeight = *no*

IF SeamDefect IS *high* THEN ImproperSeal = *high*
IF Vacuum IS *low* AND Weight IS *low* THEN ImproperSeal = *high*
IF Vacuum IS *low* AND Weight IS *normal* THEN ImproperSeal = *high*
IF Vacuum IS *normal* AND SeamDefect IS *low* THEN ImproperSeal = *low*
IF Vacuum IS *high* AND Weight is *normal* THEN ImproperSeal = *no*

IF SeamDefect IS *no* AND Weight IS *normal* AND Vacuum is *normal* THEN Good = *high*
IF SeamDefect IS *low* OR Vacuum IS *low* THEN Good = *low*
IF SeamDefect IS *high* THEN Good = *no*
IF Weight IS NOT *normal* THEN Good = *no*

Figure 6.6: Rules used to identify the classification of metal cans from primary features.

IF UnderWeight IS *high* THEN Decision = *reject*
IF OverWeight IS *high* THEN Decision = *reject*
IF ImproperSeal IS *high* THEN Decision = *reject*
IF Good IS *low* AND ImproperSeal IS *low* THEN Decision = *reject*
IF Good IS *low* AND UnderWeight IS *low* THEN Decision = *reject*
IF ImproperSeal IS *low* AND UnderWeight IS *low* THEN Decision = *reject*

Figure 6.7: Rules used to decide whether to reject cans based on object classifications.

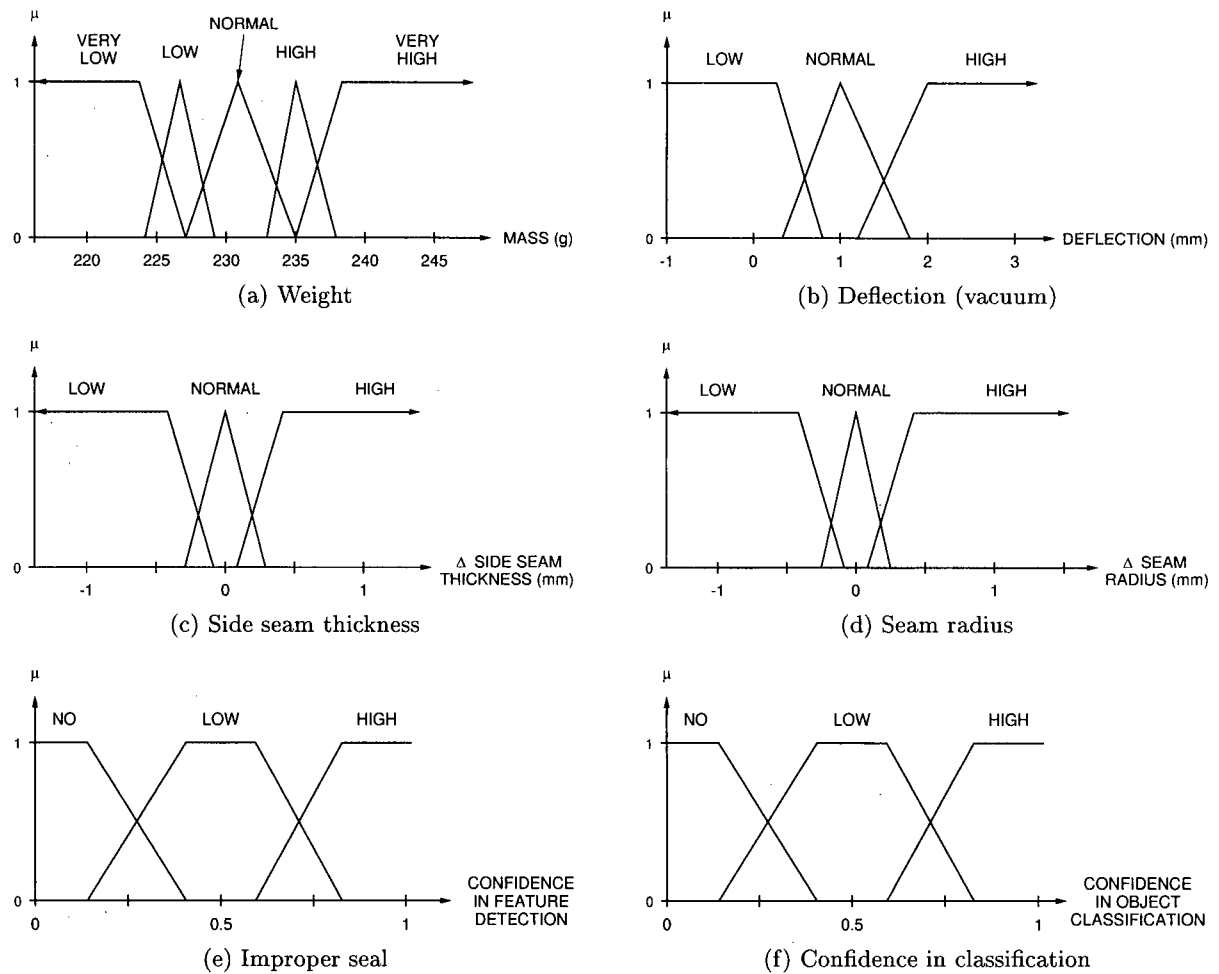


Figure 6.8: Membership functions used for classification of metal can defects.

6.1.6 Summary

To construct an industrial system, the procedure outlined in Section 5.5 is followed. In this work the object model, sensor hierarchy, and rulebase are given as examples that provide a simple introduction to the specification and construction of a multisensor system using the ELSA approach.

The can inspection problem, while simple from a modelling perspective, required the use of multiple physical cameras in combination with an ELS that fuses this information to provide a continuous image of the can side. This approach was chosen both to illustrate how such fusion would be accomplished within ELSA, but also as a practical solution to the problem. Other solutions which would minimize the number of required cameras may require the can to be rotated for a series of images — a complex and time-consuming procedure.

6.2 Herring Roe Grading

6.2.1 Background


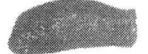

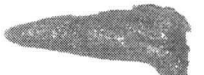


Herring roe is an important part of the B.C. economy, with an annual value of \$200 million dollars. A herring roe skein is a sac of tiny herring eggs. Two skeins are produced by each female herring. These skeins are extracted and processed for human consumption. The value of herring roe is largely influenced by the Japanese market, where it is considered a delicacy.

Being a natural product, it exhibits many non-uniform characteristics. Roe is a particularly challenging product due the large number of classifications. Each classification is dependent on the presence or absence of a number of features. Appearance and texture of the salted herring roe are the primary factors influencing price. Proper classification allows processors to offer improved value to their customers.

Currently, the process of grade classification is done manually. Herring roe is assigned a subjective grade according to aesthetic properties including colour, texture, size, and shape. Of these, all but texture are assessed visually; texture is assessed by tactile examination. The highest quality roe are light yellow in colour, stain-free, firm, over 75 mm in length, and fully formed without twists, cracks, and breaks. Heavy roe command a disproportionately higher market value. The various classifications of herring roe are presented in Table 6.1.

The roe grades are subject to change each season, due to the customer driven nature of the industry. Currently, there is no standardization of the various grade specifications. Distortions of the roe are commonly described using linguistic terms — the interpretation of which varies among expert graders. This inconsistency makes the quantification of product quality difficult.

Table 6.1: Summary of herring roe grades.

Grade No.	Grade Name	Mass (g)	Length (mm)	Description	Example
3L No. 1	Toku Toku Dai	≥ 41	> 76	Fully formed mature roe. <i>Minor</i> twists may be allowed.	
2L No. 1	Toku Dai	31–40			
Large No. 1	Dai	21–30			
Medium No. 1	Chu	16–20			
Small No. 1	Sho	10–15			
N/A	Sho Sho (pencil roe)	< 10			
2	Grade 2	N/A	> 51	Broken parts at either end.	
2-H	Light Henkei	N/A	> 76	Mature roe. Moderate to severe distortions due to air bladder, feed-sac, mishandling, etc.	
2-C	Cauliflower	N/A	N/A	Mature roe. A piece of roe that has a part extruded out from the skien, caused by a split belly or other types of damage.	
2-2	Daruma (plug roe)	N/A	N/A	Two skeins that cannot be separated.	N/A
3	Grade 3	N/A	13–51	Mature roe. Broken pieces.	N/A
3-3	Fragile roe	N/A	N/A	Mature roe. Soft texture — crumbles when pressed.	
4	Triangle roe (free roe)	N/A	N/A	Pieces of roe that are no longer contained within the skien.	
5-5	Immature roe	N/A	N/A	N/A	N/A
6	Grade 6	N/A	< 13	Mature roe. Crumbs, bit, and pieces.	N/A

6.2.2 Problem Definition/Requirements Specification

The growth of the herring roe industry has increased competition. Fisheries from other countries, particularly from Alaskan-based American companies, have begun vying for market share. This, in combination with the development of imitations made from capelin roe, have negatively affected the demand for B.C. produced herring roe. As a result of the increased competition, the consumers have begun demanding higher quality at lower prices. This places considerable pressure on the B.C. processors to improve the quality of the product through better grading, while reducing costs through increased efficiency. With the viability of current practices in question, processors have started to reassess the process and are looking towards automation as a way to realize both improved consistency and increased speed.

The Industrial Automation Laboratory at the University of British Columbia has, through a number of research projects, attempted to develop systems for the automated grading of herring roe. Since manual grading of herring roe skeins is based mainly on visual information, optical imaging has been considered as the primary sensing means for automated machine grading. A prototype herring roe grader, Figure 6.9, has been developed by Kurnianto [38, 77].



Figure 6.9: Prototype herring roe grading system.

Herring roe is a somewhat more complicated application than the previous example since each piece of roe must be assigned a grade rather than simply identified as defective. From the classifications presented in Table 6.1, the ones of primary interest for automated grading are those in the various subclassifications of Grade 1 and Grade 2 roe. The prototype system makes grading decisions based on binarized 2D profile information provided by a single CCD camera. Roe are ejected into bins at one of six gates: 3L, 2L, and Large No. 1 into the first; Medium No. 1 into the second; Small No. 1 into the third; No. 2 into the fourth; No. 2-H into the fifth; and, No. 2-C into the sixth. The other classifications are not differentiated and are allowed to fall off the end of the belt into a seventh bin.

The shape of the roe has been the most difficult to access using machine vision [35]. Human graders look for roe to be 'well formed,' or free from defects, as shown in the first row of Table 6.1. The current prototype, limited to the use of a single image without intensity information, is unable to consistently classify roe with various defects. The original system was designed as a two-classifier — separating good roe from bad — and while Grade 1 roe are identified consistently, attempts to subclassify the defective roe have met with limited success. To address these limitations, additional information is required. Three dimensional and texture information would provide the system with features that are essential to proper classification.

The design of a second generation prototype, which can integrate the additional sensor information to better distinguish roe classifications, is the aim of a new effort. The requirements of the multisensor system for the grading of herring roe skeins are as follows:

1. Accurate determination of skein length (± 1 mm).
2. Estimation of skein weight ($5\text{--}50\text{ g} \pm 0.5\text{ g}$).
3. Estimation of roe thickness as ratio of width.
4. Detection of parasite bites.
5. Detection of broken skeins (broken head or broken tail).
6. Detection of depressions ($> 12\text{ mm}^2$).
7. Detection of twists ($> 12\text{ mm}^2$).

8. Detection of proper yellow colour.
9. Detection of proper firmness of roe as an indicator of maturity.
10. Detection of bumps and curvature characteristics representative of cauliflower deformation.
11. Detection of cracks in the roe skein (> 1 mm).

6.2.3 Object Model Development

From the requirements, there are eleven salient features that can be identified as necessary for classification. These include weight, length, thickness, firmness, presence/absence of parasite bite(s), breaks, cracks, twists, depressions, cauliflower, and proper colour. Many of these can be assessed on the basis of 2D visual information. Length, breaks, cracks, cauliflower bumps, parasite bites, and proper colour are all visible from an overhead view of the roe.

Thickness, twists, and depressions require information about the three dimensional profile of the roe skien. The 3D profile is usually represented as a surface map. Due to the variability of roe, thickness is represented as a ratio between the depth of the roe (as estimated by the 3D profile) and the width of the roe at the minor axis.

Weight cannot be measured directly. There are a number of systems which can measure mass as the product moves along the conveyor; however, they tend to suffer from two problems: one is the cost of such systems, and the other, more important, is inaccurate measurement of skeins with low weight (< 10 g). As an alternative, the weight may be estimated using a linear regression model based on the peripheral length, area, and thickness of the roe [77].

Firmness is the only feature that does not lend itself to direct visual inspection. Traditionally, firmness has been assessed by handling the roe; however, this approach is not practical for an automated system. [37]. Another method, investigated in the IAL, is the use of ultrasonic echo imaging. The strength of the echo signal is directly dependent on the structure, uniformity, and firmness of the object region which generates the echo. Therefore, the echo image contains features correlated with the firmness of the roe. These features may be extracted and used as an indirect measure.

None of the primary features are atomic. Each is broken down into the various atomic components which permit the detection of the feature. The primary features and corresponding subfeatures are shown in Figure 6.10.

The primary features are combined to produce eleven different classifications. These are outlined in Table 6.2. Grade 1 roe is subclassified into six grades, according to weight. Each of these subclasses must be free of defect features, such as breaks and cracks. Additionally, Grade 1 roe must be firm, of the proper colour, and of sufficient size. If all of these criteria are satisfied, the roe is Grade 1; it is assigned to one of the subclassifications on the basis of weight.

For other grades, classification is dependent on the detection of certain distinguishing features. For example, the detection of a break will classify a piece of roe as Grade 2 provided the roe is also firm and of sufficient length.

6.2.4 Logical/Physical Sensor Selection

A logical sensor hierarchy is constructed from the object model starting with the primary features. Each of the features identified during the development of the object model is associated with an ELS which can extract the required feature. Because many of the primary features share common subfeatures, the logical sensor hierarchy, Figure 6.11, is considerably less complex than the object model. There are three physical sensors required: two CCD cameras and an ultrasonic probe; the details of each follow:

The low-level subfeatures RGB Image and 2D Profile, from which a number of other subfeatures are derived, may be provided by a single colour camera with evenly distributed, diffuse lighting. Images obtained under such conditions are presented in Figure 6.12. A JVC TK1070U colour CCD camera with a 16mm f1:1.4 lens was used.

The Echo Image subfeature is provided by a 10 MHz mechanical sector ultrasound probe. This provides input to the ELSs responsible for estimating of the firmness of a roe skein. Upon extracting the relevant image features, these are passed to the Firm ELS which uses fuzzy logic to estimate firmness.

The 3D Profile Extractor ELS utilizes a second CCD camera in combination with a structured laser light. A brief explanation of the operation of this ELS follows. The structured light was

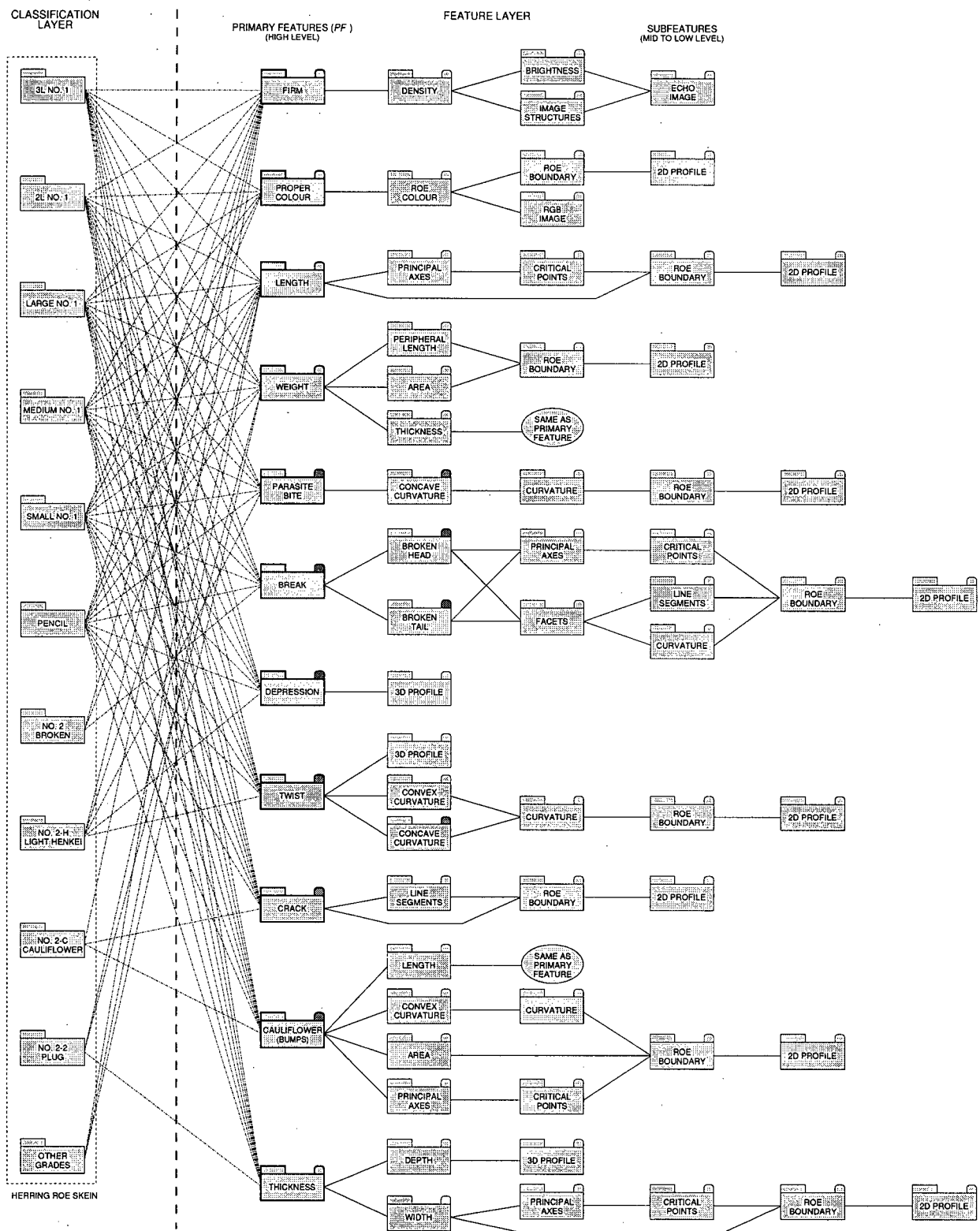


Figure 6.10: Object model for herring roe grading.

Table 6.2: Dependencies of roe classifications on primary features. Blanks indicate that the feature is not used in the assessment of the classification.

Classification	Firm	Proper Colour	Length	Weight	Parasite Bite	Break	Depression	Twist	Crack	Cauliflower	Thickness
3L-No1	Yes	Yes	Normal	Very Very Large	None	None	None	Low	None	None	Average
2L-No1	Yes	Yes	Normal	Very Large	None	None	None	Low	None	None	Average
Large-No1	Yes	Yes	Normal	Large	None	None	None	Low	None	None	Average
Medium-No1	Yes	Yes	Normal	Medium	None	None	None	Low	None	None	Average
Small-No1	Yes	Yes	Normal	Small	None	None	None	Low	None	None	Average
Pencil	Yes	Yes	Normal	Very Small	None	None	None	Low	None	None	Average
No2	Yes		Small-Normal			Yes					
No2-H	Yes		Normal		Yes		Yes	Medium-High			
No2-C	Yes								Yes	Yes	
No2-2			Normal	> Very Large							High
Other	Low-Average		< Very Small	< Small							

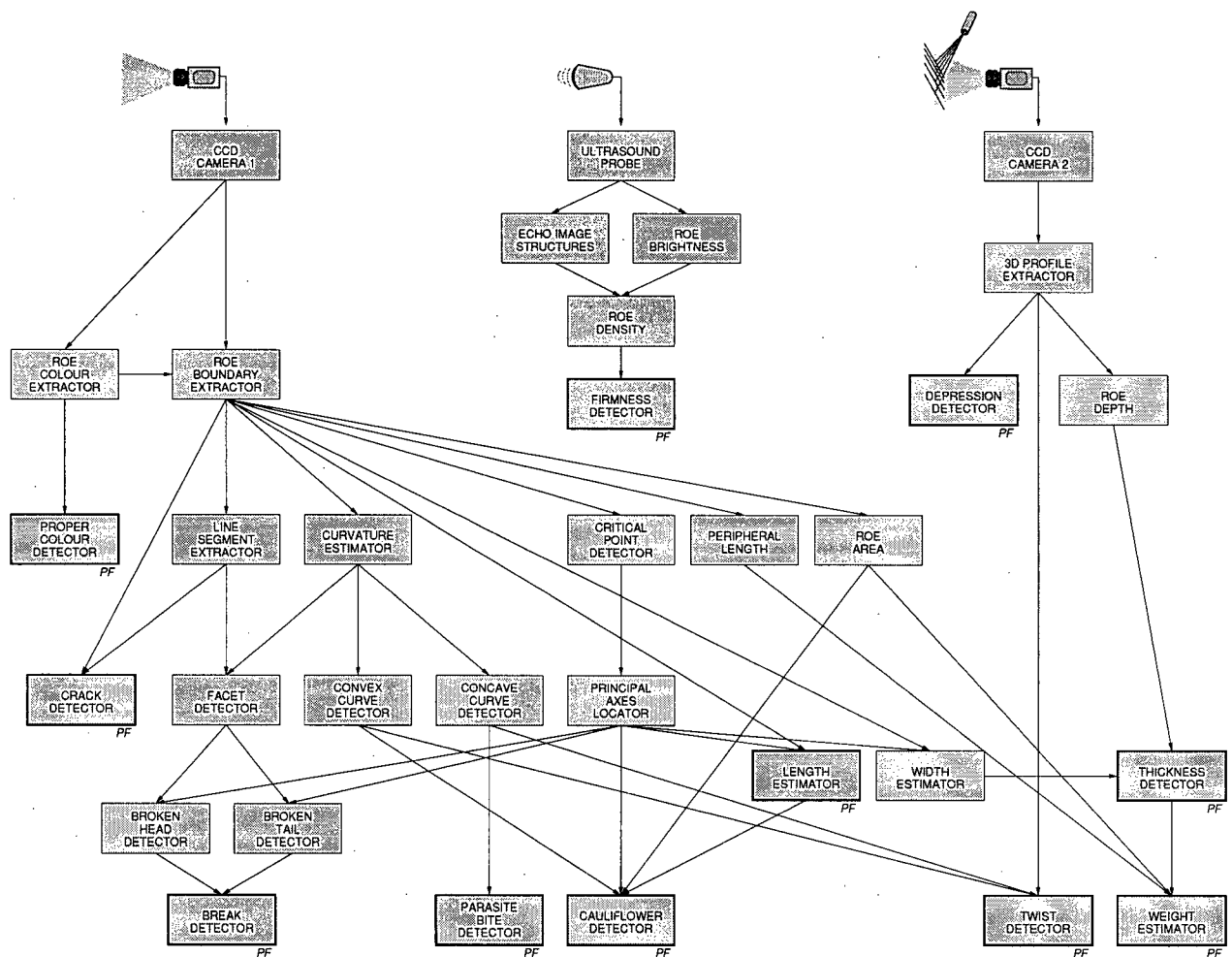
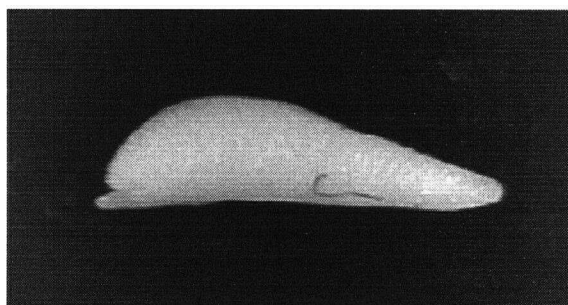
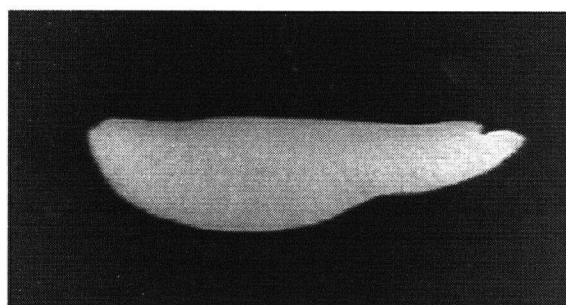


Figure 6.11: Logical sensor hierarchy for herring roe grading. Sensors which provide primary features are outlined in bold and tagged *PF*.



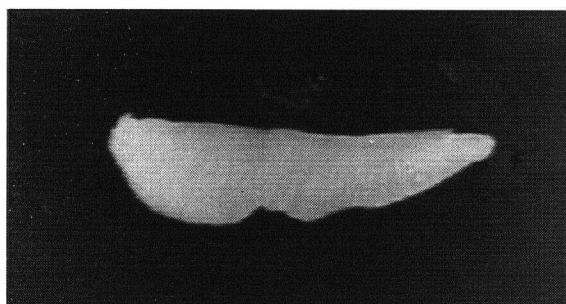
(a) Grade 1



(b) Grade 1



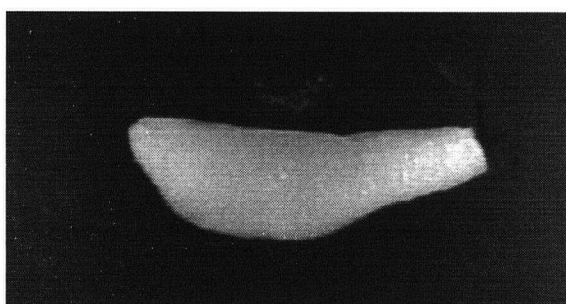
(c) Grade 2-H: twist, cracks



(d) Grade 2-H: bites from belly



(e) Grade 2-H: flat, twist



(f) Grade 2: broken tail

Figure 6.12: Examples of herring roe classification grades imaged on-line under diffuse light conditions.

mounted over the conveyor such that the band containing the eleven brightest lines was centred in the field of view of the camera. No other light source was used. Images were acquired using the red channel to maximize the brightness of the lines and minimize noise from other spectra. This setup is illustrated by Figure 6.13. The exact spacing between the rays, and the resultant projected lines, is detailed in Table 6.3. With this arrangement, one pixel is 0.392 mm square and $\Delta\theta_{ray} = 1.5316^\circ$, where $\Delta\theta_{ray}$ is the angle between the rays cast from the structured light.

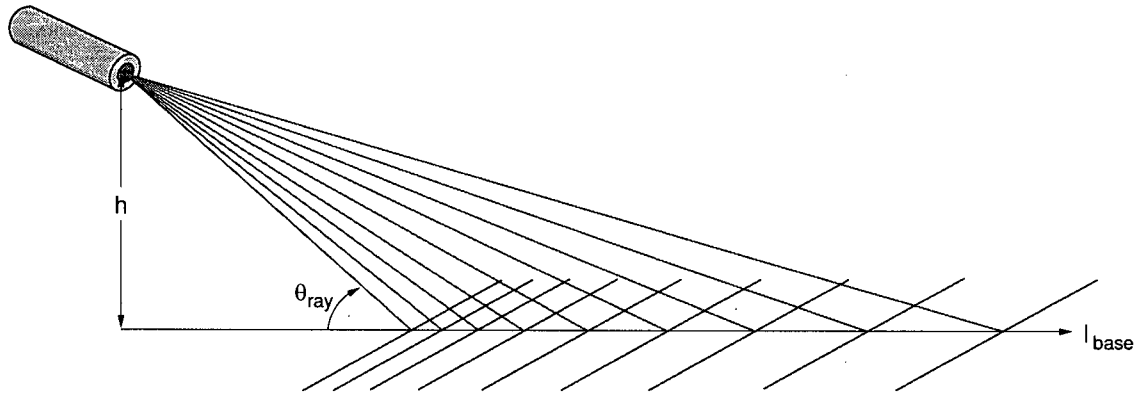


Figure 6.13: Geometry of structured light used for acquisition of 3D features.

A 3D Profile Extractor ELS reconstructs the 3D profile using this knowledge about the pixel size and the fixed spacing of the light rays. For each pixel corresponding to a projected light ray, the height of an object may be reconstructed using Equation (6.1).

$$\begin{aligned} h_{pixel} &= \delta_{pixel} \cdot \tan \theta_{ray} && [\text{pixels}] \\ &= 0.392 \cdot \delta_{pixel} \cdot \tan \theta_{ray} && [\text{mm}] \end{aligned} \quad (6.1)$$

where, h_{pixel} is the height of the object at the given pixel location. For a given ray, δ_{pixel} is the number of pixels the projected line has displaced on the object with respect to the undisplaced (straight line) projection on the conveyor and θ_{ray} is the ray angle as in Table 6.3. The relationship between these variables is shown in Figure 6.14.

Examples of roe skeins imaged under the structured light are presented in Figure 6.15.

Table 6.3: Calculation of structured light geometry. Based on $h = 160\text{mm}$.

Ray Number	l_{base} (mm)	θ_{ray} ($^{\circ}$)
1	155.0	45.91
2	163.5	44.38
3	172.8	42.80
4	182.3	41.27
5	192.5	39.73
6	203.0	38.24
7	214.5	36.72
8	227.0	35.18
9	240.0	33.69
10	254.5	32.16
11	270.5	30.60
12	287.8	29.07
13	307.0	27.53

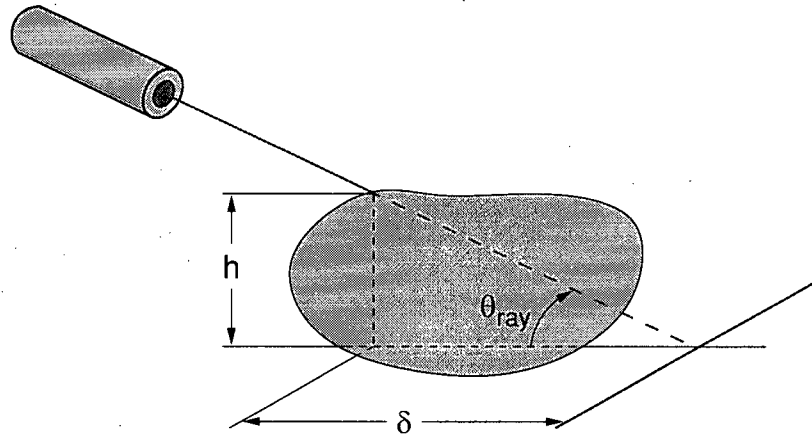


Figure 6.14: Basic geometry for reconstruction of 3D profile information using structured light.

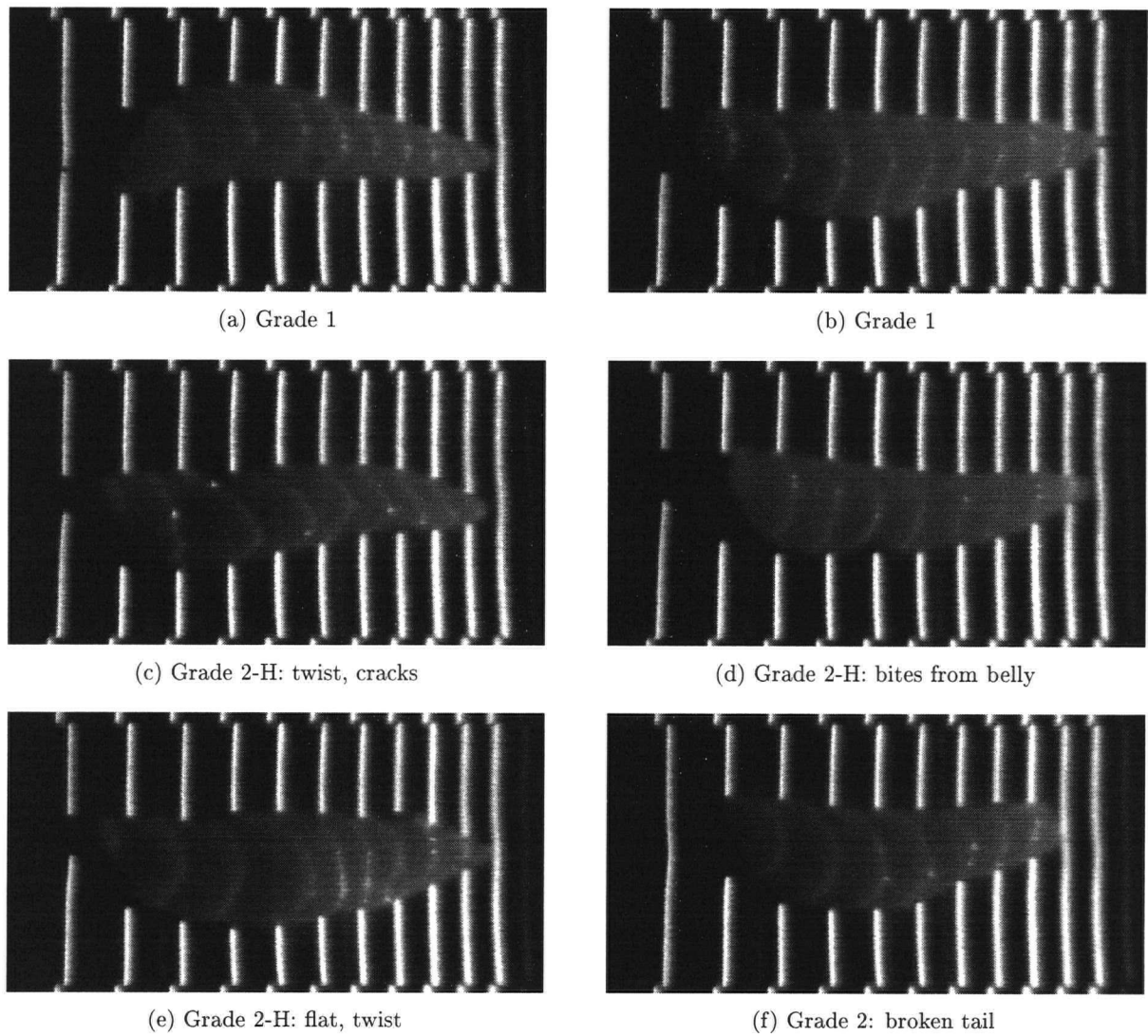


Figure 6.15: Example of herring roe classification grades imaged on-line under structured light conditions.

6.2.5 Rulebase Definition

The rulebase generation follows from the object model. The object classifications outlined in 6.2.3 are used as the basis for the classification rules, Figure 6.16. The confidence membership function is used to express the confidence in the detection of the proper firmness, proper colour, breaks, cauliflower deformities, parasite bites, depressions, and cracks. The other features: length, weight, thickness, and twist, utilize specific membership functions.

```

IF Firm IS high AND Length IS normal AND ProperColour IS high AND Weight IS very very large AND ParasiteBite IS no AND Break IS no AND Depression IS no AND Twist IS no AND Crack IS no AND Cauliflower IS no AND Thickness IS normal THEN 3L-No1 = high
IF Firm IS high AND Length IS normal AND ProperColour IS high AND Weight IS very very large AND ParasiteBite IS no AND Break IS no AND Depression IS no AND Twist IS no AND Crack IS no AND Cauliflower IS low AND Thickness IS normal THEN 3L-No1 = high
IF Firm IS low OR ProperColour IS low OR ParasiteBite IS low OR Break IS low OR Depression IS low OR Twist IS medium OR Crack IS low OR Cauliflower IS low AND Weight IS very very large THEN 3L-No1 = low
IF Firm IS no OR ProperColour IS no OR ParasiteBite IS high OR Break IS high OR Depression IS high OR Twist IS high OR Crack IS high OR Cauliflower IS high AND Weight IS very very large THEN 3L-No1 = no
:
:
IF Firm IS high AND Length IS small AND Break IS high THEN No2 = high
IF Firm IS high AND Length IS normal AND Break IS high THEN No2 = high
IF Firm IS high AND Length IS small AND Break IS average THEN No2 = low
IF Firm IS high AND Length IS normal AND Break IS average THEN No2 = low
IF Break IS average THEN No2 = no

IF Firm IS high AND Length IS normal AND ParasiteBite IS high THEN No2-H = high
IF Firm IS average AND Length IS normal AND ParasiteBite IS high THEN No2-H = high
IF Firm IS high AND Length IS normal AND ParasiteBite IS average THEN No2-H = low
IF Firm IS high AND Length IS normal AND Depression IS high THEN No2-H = high
IF Firm IS average AND Length IS normal AND Depression IS high THEN No2-H = high
IF Firm IS high AND Length IS normal AND Depression IS average THEN No2-H = low
IF Firm IS high AND Length IS normal AND Twist IS high THEN No2-H = high
IF Firm IS average AND Length IS normal AND Twist IS high THEN No2-H = high
IF Firm IS high AND Length IS normal AND Twist IS medium THEN No2-H = high
IF Firm IS high AND Length IS normal AND Twist IS low THEN No2-H = no
IF ParasiteBite IS low AND Depression IS low AND Twist IS low THEN No2-H = no

IF Firm IS high AND Crack IS high THEN No2-C = high
IF Firm IS high AND Crack IS average THEN No2-C = low
IF Firm IS high AND Cauliflower IS high THEN No2-C = high
IF Firm IS high AND Cauliflower IS average THEN No2-C = low
IF Crack IS average AND Cauliflower IS average THEN No2-C = low
IF Crack IS low AND Cauliflower IS low THEN No2-C = no

IF Firm IS low THEN Unclassified = high
IF Firm IS average THEN Unclassified = low
IF Length IS very small THEN Unclassified = high
IF Length IS very very small THEN Unclassified = high
IF Length IS very small AND Weight IS small THEN Unclassified = high
IF Length IS very very small AND Weight IS very small THEN Unclassified = high
IF Length IS small AND Weight IS small THEN Unclassified = low

```

Figure 6.16: Rules used to identify herring roe grades from primary features. For clarity, a number a rules used for classifying Grade 1 roe have been removed.

Once the roe have been classified, a decision is made about which bin it should be ejected into. The classifications are segregated using the same apparatus used by the prototype grading system. Bin 1 accepts Grade 1 Large, 2L, and 3L; Bin 2 accepts Grade 1 Medium; Bin 3 accepts Grade 1 Small; Bin 4 accepts Grade 2; Bin 5 accepts Grade 2-H; Bin 6 accepts Grade 2-C; and Bin 7 accepts all other grades and unclassified roe which fall off the end of the conveyor. Figure 6.17 presents the rules which are used to infer this decision. The fuzzy membership functions associated with these rules are shown in Figure 6.18.

IF Grade1-3L IS <i>high</i> THEN Decision = <i>bin1</i>
IF Grade1-2L IS <i>high</i> THEN Decision = <i>bin1</i>
IF Grade1-Large IS <i>high</i> THEN Decision = <i>bin1</i>
IF Grade1-Medium IS <i>high</i> THEN Decision = <i>bin2</i>
IF Grade1-Small IS <i>high</i> THEN Decision = <i>bin3</i>
IF Grade1-Pencil IS <i>high</i> THEN Decision = <i>bin7</i>
IF Grade2 IS <i>high</i> THEN Decision = <i>bin4</i>
IF Grade2 IS <i>low</i> THEN Decision = <i>bin4</i>
IF Grade2-C IS <i>high</i> THEN Decision = <i>bin5</i>
IF Grade2-C IS <i>low</i> THEN Decision = <i>bin5</i>
IF Grade2-H IS <i>high</i> THEN Decision = <i>bin6</i>
IF Grade2-H IS <i>low</i> THEN Decision = <i>bin6</i>
IF Unclassified IS <i>high</i> THEN Decision = <i>bin7</i>
IF Unclassified IS <i>low</i> THEN Decision = <i>bin7</i>

Figure 6.17: Rules used to determine decisions about how roe should be handled based on object classifications.

6.2.6 Summary

The herring roe grading application is considerably more complex than the previous example, metal can inspection. The non-uniform nature of the product and subjective classification criteria significantly increase the complexity of the object model. Despite this, the ELSA approach serves to guide the user through the development process in a systematic manner. This ensures that the final design satisfies the functional requirements, but may also be augmented or modified with a minimal amount of disturbance to the system as a whole.

It is interesting to note that despite the increased number of classifications and features, the required number of physical sensors is less than half of what was required by the previous example. By building the ELS hierarchy from the object model, the redundancy in sensing requirements becomes obvious. Again, the system would be implemented following the procedure outlined in

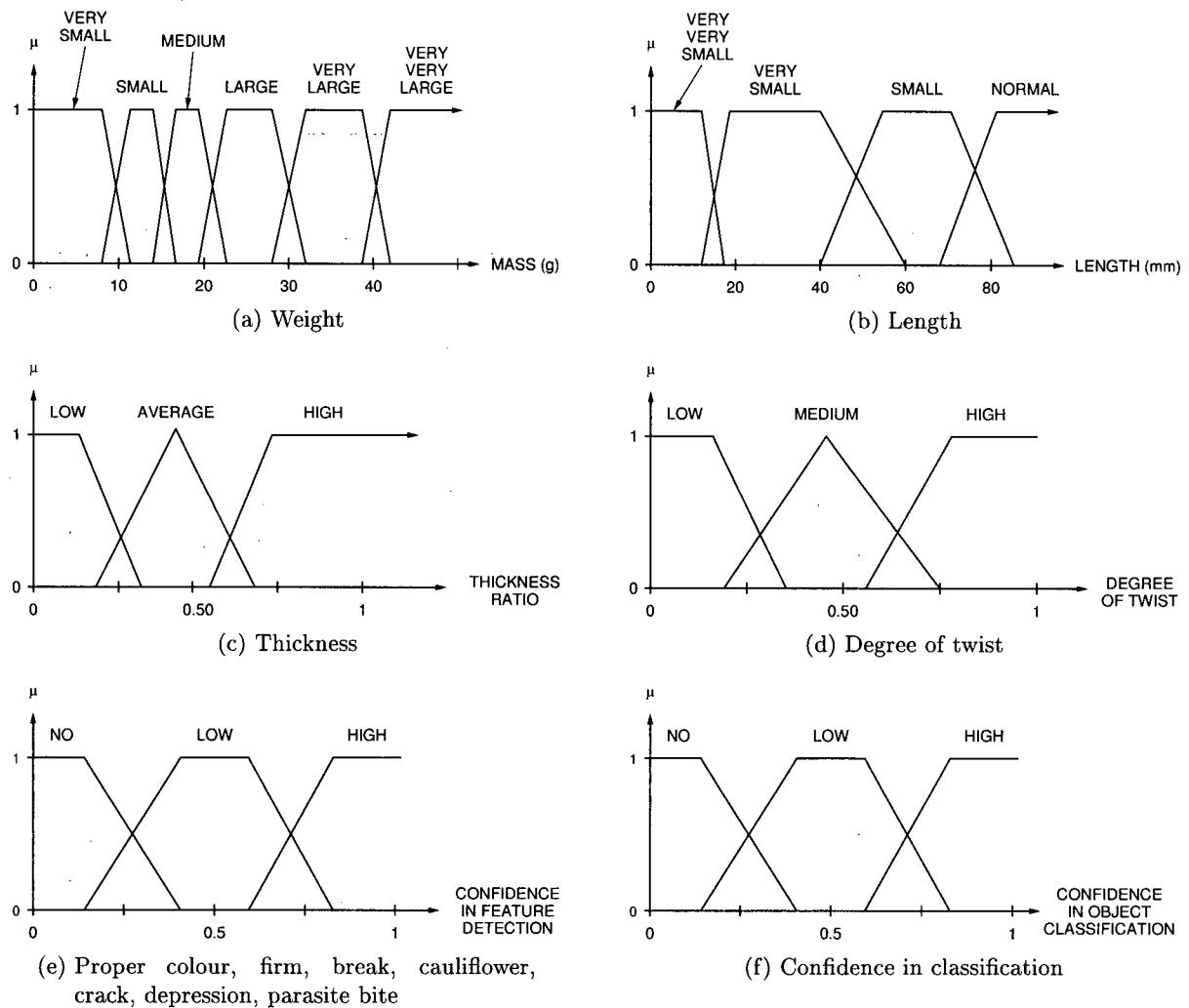


Figure 6.18: Membership functions used for classification of herring roe grades.

Section 5.5.

6.3 Discussion

Through these applications, the advantages of the ELSA approach to system design are demonstrated. By formalizing the design process, a system can be designed to meet the specified functional requirements in a systematic and comprehensible way. Each stage involves the extraction and utilization of the user's (e.g. a quality assurance engineer's) expert knowledge about the process and desired outcomes. Specification of the requirements leads to the identification of primary features

and object classifications. These are expanded into subfeatures. The features themselves determine the algorithms (encapsulated logical sensors) and physical sensors that are required by the system. Decisions are inferred directly from the object classifications.

Perhaps the most challenging aspect of ELSA is the construction of logical sensors that are not available from the library. This requires some knowledge of signal processing and the internal workings of the ELS model that the industrial user may not possess. In these instances, the user would be advised to define the specification of the sensor using their expert knowledge and then contract the construction of the sensor to a technical expert. The specification process effectively separates the expert domain knowledge from the technical programming knowledge required to develop an ELS. Once such sensors are defined (and consequently available from the library), the construction, modification, and comprehension of the ELSA-based system is more tractable for a non-technical domain expert.

The object models and sensor hierarchies presented herein should not be considered as *the* solution. The selection of different features and sensor combinations may yield systems with similar or better performance. Systems may be designed to take advantage of certain equipment or in-house expertise. The design of the herring roe system, for example, is in part dependent on the familiarity with, and the availability of, machine vision systems and software in the IAL.

Nor are these systems static. Should needs dictate, the object model and/or sensor hierarchies may be modified to meet new conditions. For example, should a cost-effective system be developed for physically measuring the skein weight, this may replace the vision-based weight estimation ELS to result in an ELS hierarchy for weight much like that presented for can inspection.

The structure of the architecture ensures that should additional capabilities be desired (e.g. for can inspection: the inspection of stamp codes, lid ring profiles, or detection of pin holes in the can body), they may be added without affecting the existing components. The object model is expanded to include the additional features and/or classifications. Any required logical sensors are added to the ELS hierarchy. This is accomplished without disturbing the remainder of the system.

Chapter 7

Concluding Remarks

This work presented a methodology for the design and construction of multisensor integration systems for industrial applications, with particular emphasis on non-uniform product inspection and grading. Specifically, the following research objectives were considered:

1. To specify a data representation that can represent non-uniform objects in a simple, flexible, and understandable way.
2. To design the data representation such that it can be used to guide the construction of the system.
3. To provide a modular and scalable architecture for intelligent industrial sensing applications.
4. To specify an encapsulation of physical devices and processing algorithms.
5. To provide a robust exception handling mechanism to ensure the reliability of the system.
6. To ensure that the architecture is applicable to a broad range of industrial applications.

Each of these objectives was considered and developed to some degree. As specified, the ELSA object model provides both a guide for system construction and represents deviations of non-uniform objects. An Extended Logical Sensor model encapsulates sensing devices and algorithms. The ELS and the object model together provide a basis for a modular and scalable architecture that is particularly applicable to a variety of industrial grading applications. An exception handling

mechanism has been proposed. However, a substantial amount of work still remains to develop a complete industrial version of ELSA.

7.1 Summary and Conclusions

ELSA is a multisensor integration architecture for industrial tasks. It is also, based upon the object model, a methodology for the construction of such a system. ELSA was developed to provide an organized approach to the development of industrial-based sensor systems. It addresses the need for scalable, modular, and structured sensor systems, replacing current ad hoc approaches. The construction methodology enables domain experts, who lack signal processing knowledge, to design and understand a sensor system for their particular application.

To achieve this, ELSA is comprised of a number of different components. Extended Logical Sensors are presented as an improvement to the existing LS and ILS specifications. This improvement is realized by strongly encapsulating the ELS. The ELS may be polled by other sensors to determine its capabilities and request changes in the performance of the ELS, but its internal operation is hidden. Replacement sensors need only provide the same form of output. Other components, such as the Exception Handling Mechanism and the Integration Controller, serve to enhance the robustness and functionality of the architecture.

The object model used by ELSA is particularly suited to the representation of non-uniform products, or any object for which classification is desired. Objects are described in terms of their primary or distinguishing features. Primary features may be a composite of subfeatures. Objects are classified by using fuzzy membership functions to express how the primary features combine for each classification. The organization of the sensor system and the definition of the rulebase is driven by the object model.

Logical sensors are chosen to provide each of the features defined by the object model; this in turn determines what physical sensors are required by the system. The classification layer of the object model directly specifies how primary features are combined to determine object classifications. To demonstrate these concepts, ELSA was applied to the problems of metal can inspection and herring roe grading.

The design and implementation of an ELS requires signal processing and programming knowl-

edge that an industrial user may not possess. Although this limits the ability of such a user to fully construct a system, it may be completely specified. This is because ELSA effectively separates the domain knowledge from the detailed sensor knowledge. If necessary, a technical expert may be consulted to develop the required ELS(s). Once a library of standard logical sensors is established for a set of applications, a system may be constructed without an in-depth understanding of the internal workings of each ELS. This makes ELSA particularly suitable for industrial users who wish to construct, modify, and maintain industrial multisensor systems.

7.2 Recommendations

This thesis provides the groundwork for a much larger and more complete system. It is now necessary to further develop the ideas presented herein — completing the implementation of what has been specified, and extending this specification to include new capabilities.

A library of Extended Logical Sensors should be constructed that is suitable for a variety of inspection and grading tasks. This will assist in the development of ELSA-based systems for applications such as the grading of herring roe, potatoes, blueberries, and other produce.

There are many extensions that could increase the user friendliness and automation of the system specification and construction. These would serve to further remove the industrial user from the technical details of system design, promoting better understanding and allowing the user to focus on the process.

Most of the components of ELSA have been designed with the automation of the system construction in mind. This includes the object model development, rulebase generation, logical/physical sensor selection, the Integration Controller, Validation and Diagnostics modules (exception handling). This should enable a variety of extensions to be implemented with ease.

One such extension is the implementation of an expert system that could be used to further guide the selection of physical sensors and ELSs. This could work towards an optimal selection of sensor components based on user-defined constraints such as system cost, speed, accuracy, etc. This would be particularly useful for ensuring that the designed system can operate at line speeds and for selecting appropriate sensor combinations to provide robustness through redundancy. The expert system could also serve to ensure the completeness and uniqueness of a particular system

configuration.

The membership functions contained in the rulebase could be automatically generated from the object model. For unquantifiable features, use of a confidence membership would be used; for others, the user would be prompted for the universe of discourse (range of expected values) and linguistic variables describing classifications over the universe. Once generated, the system could automatically tune and refine the membership functions for optimum performance. This would reduce the need for users to have an in-depth understanding of fuzzy logic. Expert users should be still able to by-pass the system, enabling direct definition and fine-tuning of the membership functions.

For very complex objects, it may be useful to allow a variation of the object model presented herein. The approach would be similar except that the object model would be hierarchical, further increasing the compactness and efficiency of the feature-based object model. It would work by placing defective classifications on the first level and 'good' classifications on another. If the object does not present any of the features that would classify it as defective (each classification represented by a minimal set of features), then the object could then be classified as good. Subclassification of the good category, based on features such as size, weight, and colour could then proceed without the need to determine if a defect is present. The inference mechanism could also be further refined by allowing rules to be weighted. This would allow rules, and the corresponding features, to be given different emphasis.

Further work is required to extend ELSA to control applications. One approach to this may be the concept of a Logical Actuator (LA). Control decisions made by the Inference Engine would be passed to a LA hierarchy where directives are converted into actions. The logical actuators thus serve as an interface between the high-level decision making system and the low-level process machinery. In this sense, a LA is an analogue to a LS. A similar idea, a combined Logical Sensor/Actuator (LSA) presented by Budenske and Gini [78]. By encapsulating the physical actuators, drivers, and planning algorithms, they may be altered without affecting the Inference Engine.

This concept could be extended by combining the logical sensor and logical actuator into a common model — a logical device. The use of intelligent software agents could further encapsulate this concept. As an extension of the object-oriented nature of the ELS and logical actuator hier-

archies, agents may further increase the openness and flexibility of the system. Through the use of software agents, each sensor, algorithm, controller, actuator, etc. becomes a separate module which may interact with other modules through a specified protocol. Dependencies on particular hardware configurations and software algorithms are further reduced, if not eliminated. Of course, any serious effort to implement better control will also have to consider the problems and issues that arise when dealing with real-time control systems.

References

- [1] E. R. Davies, *Machine Vision: Theory, Algorithms, Practicalities*. Signal Processing and its Applications, San Diego, CA: Academic Press, 2nd ed., 1997.
- [2] D. A. Beatty, "2D contour shape analysis for automated herring roe quality grading by computer vision," Master's thesis, Department of Computer Science, University of British Columbia, Vancouver, B.C. V6T 1Z4, Dec. 1993.
- [3] B. D. Allin, "Analysis of the industrial automation of a food processing quality assurance work-cell," Master's thesis, Department of Mechanical Engineering, University of British Columbia, Vancouver, B.C. V6T 1Z4, Apr. 1998.
- [4] S. Gunasekaran, "Computer vision technology for food quality assurance," *Trends in Food Science & Technology*, vol. 7, pp. 245-256, 1996.
- [5] J. M. Fildes and A. Cinar, "Sensor fusion and intelligent control for food processing," in *Food Processing Automation II: Proceedings of the 1992 Conference*, (Lexington, KY), pp. 65-72, FPEI, May 4-6 1992.
- [6] R. C. Luo and M. G. Kay, "Data fusion and sensor integration: State-of-the-art 1990s," in *Data Fusion in Robotics and Machine Intelligence* (M. A. Abidi and R. C. Gonzalez, eds.), pp. 7-135, San Diego, CA: Academic Press, 1992.
- [7] R. A. Brooks, "A layered intelligent control system for a mobile robot," in *The Third International Symposium on Robotics Research* (O. D. Faugeras and G. Giralt, eds.), (Gouvieux, France), pp. 365-372, 1986.
- [8] R. R. Murphy and R. C. Arkin, "SFX: an architecture for action-oriented sensor fusion," in *Proceedings of the 1992 IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 2, (Raleigh, NC), pp. 1079-1086, IEEE/RSJ, July 7-10 1992.
- [9] R. R. Murphy, "Biological and cognitive foundations of intelligent sensor fusion," *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 26, pp. 42-51, Jan. 1996.
- [10] T. G. R. Bower, "The evolution of sensory systems," in *Perception: Essays in Honor of James J. Gibson* (R. B. MacLeod and H. L. Pick Jr., eds.), pp. 141-153, Ithaca, NY: Cornell University Press, 1974.
- [11] S. Lee, "Sensor fusion and planning with perception-action network," in *Proceedings of the 1996 IEEE/SICE/RSJ International Conference on Multisensor Fusion and Integration for Intelligent Systems*, (Washington, D.C.), pp. 687-696, IEEE/SICE/RSJ, Dec. 8-11 1996.

- [12] S. Lee and S. Ro, "Uncertainty self-management with perception net based geometric data fusion," in *Proceedings of the 1997 IEEE International Conference on Robotics and Automation*, (Albuquerque, NM), pp. 2075–2081, IEEE, 1997.
- [13] B. A. Draper, A. R. Hanson, S. Buluswar, and E. M. Riseman, "Information acquisition and fusion in the mobile perception laboratory," in *Proceedings of the SPIE - Signal Processing, Sensor Fusion, and Target Recognition VI*, vol. 2059, pp. 175–187, SPIE, 1993.
- [14] S. A. Shafer, A. Stentz, and C. C. Thorpe, "An architecture for sensor fusion in a mobile robot," in *Proceedings of the IEEE International Conference on Robotics and Automation*, (San Francisco, CA), pp. 2002–2011, IEEE, 1986.
- [15] S. S. Iyengar, D. N. Jayasimha, and D. S. Nadig, "A versatile architecture for the distributed sensor integration problem," *IEEE Transactions on Computers*, vol. 43, pp. 175–185, Feb. 1994.
- [16] S. S. Iyengar, L. Prasad, and M. Min, *Advances in Distributed Sensor Technology*. Environmental and Intelligent Manufacturing Systems Series, Upper Saddle River, NJ: Prentice Hall PTR, 1995.
- [17] L. A. Klein, *Sensor and Data Fusion Concepts and Applications*, vol. TT 14 of *Tutorial Texts in Optical Engineering*. Bellingham, Washington: SPIE Optical Engineering Press, 1993.
- [18] T. Queeney and E. Woods, "A generic architecture for real-time multisensor fusion tracking algorithm development and evaluation," in *Proceedings of the SPIE - Signal Processing, Sensor Fusion, and Target Recognition VII*, vol. 2355, pp. 33–42, SPIE, 1994.
- [19] I. Alarcón, P. Rodríguez-Marín, L. B. Almeida, R. Sanz, L. Fontaine, P. Gómez, X. Alamán, P. Nordin, H. Bejder, and E. de Pablo, "Heterogeneous integration architecture for intelligent control systems," *Intelligent Systems Engineering Journal*, vol. 3, pp. 138–152, Autumn 1994.
- [20] T. C. Henderson and E. Shilcrat, "Logical sensor systems," *Journal of Robotic Systems*, vol. 1, no. 2, pp. 169–193, 1984.
- [21] T. C. Henderson, C. Hansen, and B. Bhanu, "The specification of distributed sensing and control," *Journal of Robotic Systems*, vol. 2, no. 4, pp. 387–396, 1985.
- [22] G. A. Weller, F. C. A. Groen, and L. O. Hertzberger, "A sensor processing model incorporating error detection and recovery," in *Traditional and Non-Traditional Robotic Sensors* (T. C. Henderson, ed.), vol. F 63, pp. 351–363, Berlin: Springer-Verlag, 1990.
- [23] F. Groen, P. Antonissen, and G. Weller, "Model based robot vision by extending the logical sensor concept," in *1993 IEEE Instrumentation and Measurement Technology Conference*, (Irvine, CA), pp. 584–588, IEEE, 1993.
- [24] M. Dekhil, T. M. Sobh, and A. A. Efros, "Commanding sensors and controlling indoor autonomous mobile robots," in *Proceedings of the 1996 IEEE International Conference on Control Applications*, (Dearborn, MI), pp. 199–204, IEEE, Sept. 15–18 1996.
- [25] M. Dekhil and T. C. Henderson, "Instrumented sensor systems," in *Proceedings of the 1996 IEEE/SICE/RSJ International Conference on Multisensor Fusion and Integration for Intelligent Systems*, (Washington, D.C.), pp. 193–200, IEEE/SICE/RSJ, Dec. 8–11 1996.

- [26] M. Dekhil and T. C. Henderson, "Instrumented sensor system – practice," Tech. Rep. UUCS-97-014, University of Utah, Salt Lake City, UT 84112, Mar. 1997.
- [27] M. Dekhil and T. C. Henderson, "Instrumented sensor system architecture," *The International Journal of Robotics Research*, vol. 17, no. 4, pp. 402–417, 1998.
- [28] R. Ohba, ed., *Intelligent Sensor Technology*. Chichester, England: John Wiley & Sons, 1992.
- [29] IEEE, *P1451.1 Draft Standard for a Smart Transducer Interface for Sensors and Actuators — Network Capable Application Processor (NCAP) Information Model*, D1.83 ed., Dec. 1996.
- [30] IEEE, *P1451.2 Draft Standard for a Smart Transducer Interface for Sensors and Actuators — Transducer to Microprocessor Communication Protocols and Transducer Electronic Data Sheet (TEDS) Formats*, D3.05 ed., Aug. 1997.
- [31] D. A. Luzuriaga, M. O. Balaban, and S. Yeralan, "Analysis of visual quality attributes of white shrimp by machine vision," *Journal of Food Science*, vol. 62, no. 1, pp. 113–118, 1997.
- [32] P. Jia, M. D. Evans, and S. R. Ghate, "Catfish feature identification via computer vision," *Transactions of the ASAE*, vol. 39, pp. 1923–1931, Sep./Oct. 1996.
- [33] W. Daley, R. Carey, and C. Thompson, "Poultry grading/inspection using color imaging," in *Proceedings of the SPIE - Machine Vision Applications in Industrial Inspection*, vol. 1907, pp. 124–132, SPIE, 1993.
- [34] J. Calpe, F. Pla, J. Monfort, P. Díaz, and J. C. Boada, "Robust low-cost vision system for fruit grading," in *Proceedings of the 1996 8th Mediterranean Electrotechnical Conference*, vol. 3, (Bari, Italy), pp. 1710–1713, IEEE, 1996.
- [35] L. X. Cao, C. W. de Silva, and R. G. Gosine, "A knowledge-based fuzzy classification system for herring roe grading," in *Proceedings of the Winter Annual Meeting on Intelligent Control Systems*, vol. DSC-48, (New York), pp. 47–56, ASME, 1993.
- [36] A. Beatty, R. G. Gosine, and C. W. de Silva, "Recent developments in the application of computer vision for automated herring roe assessment," in *Proceedings of the IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, vol. 2, (Victoria, Canada), pp. 698–701, IEEE, May 19–21 1993.
- [37] C. W. de Silva, L. B. Gamage, and R. G. Gosine, "An intelligent firmness sensor for an automated herring roe grader," *International Journal of Intelligent Automation and Soft Computing*, vol. 1, no. 1, pp. 99–114, 1995.
- [38] E. A. Croft, C. W. de Silva, and S. Kurnianto, "Sensor technology integration in an intelligent machine for herring roe grading," *IEEE/ASME Transactions on Mechatronics*, vol. 1, pp. 204–215, Sept. 1996.
- [39] P. H. Heinemann, N. P. Pathare, and C. T. Morrow, "An automated inspection station for machine-vision grading of potatoes," *Machine Vision and Applications*, vol. 9, pp. 14–19, 1996.
- [40] J. H. Dan, D. M. Yoon, and M. K. Kang, "Features for automatic surface inspection," in *Proceedings of the SPIE - Machine Vision Applications in Industrial Inspection*, vol. 1907, pp. 114–123, SPIE, 1993.

- [41] G. Brown, P. Forte, R. Malyan, and P. Barnwell, "Object oriented recognition for automatic inspection," in *Proceedings of the SPIE - Machine Vision Applications in Industrial Inspection II*, vol. 2183, (San Jose, CA), pp. 68–80, SPIE, 1994.
- [42] M. A. O'Dor, "Identification of salmon can-filling defects using machine vision," Master's thesis, Department of Mechanical Engineering, University of British Columbia, Vancouver, B.C. V6T 1Z4, Mar. 1998.
- [43] Dipix Technologies Inc., "QualiVision product information," 1998. <http://www.dipix.com/vissys/vissset.htm>.
- [44] "Flexible systems for trimming and portioning," *World Fishing*, pp. 13–14, June 1997.
- [45] E. A. Croft, "Personal communication and plant tour of Lullebelle Foods Ltd.," June 1996.
- [46] Key Technology, Inc., "Product catalog," 1995.
- [47] ANSI/ASME, *Measurement Uncertainty, Part I*, PTC 19.1 ed., 1986.
- [48] H. W. Coleman and W. G. Steele Jr., *Experimentation and Uncertainty Analysis for Engineers*. New York: John Wiley & Sons, 1989.
- [49] A. J. Wheeler and A. R. Ganji, *Introduction to Engineering Experimentation*. Englewood Cliffs, NJ: Prentice Hall, 1996.
- [50] P. Suetens, P. Fua, and A. J. Hanson, "Computational strategies for object recognition," *ACM Computing Surveys*, vol. 24, pp. 5–61, Mar. 1992.
- [51] S. C. Zhu and A. L. Yuille, "Forms: a flexible object recognition and modelling system," in *Proceedings of the 5th IEEE International Conference on Computer Vision*, (Cambridge, MA), pp. 465–472, IEEE, 1995.
- [52] L. Stark, K. Bowyer, A. Hoover, and D. B. Goldgof, "Recognizing object function through reasoning about partial shape descriptions and dynamic physical properties," *Proceedings of the IEEE*, vol. 84, pp. 1640–1656, Nov. 1996.
- [53] A. Z. Kouzani, F. He, and K. Sammut, "Constructing a fuzzy grammar for syntactic face detection," in *Proceedings of the 1996 IEEE International Conference on Systems, Man, and Cybernetics*, vol. 2, (Beijing, China), pp. 1156–1161, IEEE, 1996.
- [54] Z. Luo and C.-H. Wu, "A unit decomposition technique using fuzzy logic for real-time handwritten character recognition," *IEEE Transactions on Industrial Electronics*, vol. 44, pp. 840–847, Dec. 1997.
- [55] H.-M. Lee and C.-W. Huang, "Fuzzy feature extraction on handwritten chinese characters," in *Proceedings of the 1994 IEEE International Conference on Fuzzy Systems*, vol. 3, (Orlando, FL), pp. 1809–1814, IEEE, 1994.
- [56] I. Biederman, "Recognition-by-components: A theory of human image understanding," *Psychological Review*, vol. 94, no. 2, pp. 115–147, 1987.
- [57] P. Havaladar, G. Medioni, and F. Stein, "Perceptual grouping for generic recognition," *International Journal of Computer Vision*, vol. 20, no. 1/2, pp. 59–80, 1996.

- [58] F. Tomita and S. Tsuji, *Computer Analysis of Visual Textures*. Norwell, Massachusetts: Kluwer Academic Publishers, 1990.
- [59] G. K. Lang and P. Seitz, "Robust classification of arbitrary object classes based on hierarchical spatial feature-matching," *Machine Vision and Applications*, vol. 10, pp. 123-135, 1997.
- [60] D. Cho and Y. J. Bae, "Fuzzy-set based feature extraction for objects of various shapes and appearances," in *Proceedings of the 1996 IEEE International Conference on Image Processing*, vol. 2, (Los Alamitos, CA), pp. 983-986, IEEE, 1996.
- [61] L. A. Zadeh, "Fuzzy logic = computing with words," *IEEE Transactions on Fuzzy Systems*, vol. 4, pp. 103-111, May 1996.
- [62] J. H. Connell and M. Brady, "Generating and generalizing models of visual objects," *Artificial Intelligence*, vol. 31, pp. 159-183, 1987.
- [63] L. A. Zadeh, "Outline of a new approach to the analysis of complex systems and decision processes," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 3, pp. 28-44, Jan. 1973.
- [64] C. W. de Silva, *Intelligent Control: Fuzzy Logic Applications*. Boca Raton, Florida: CRC Press, 1995.
- [65] J. Canny, "A computational approach to edge detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 8, no. 6, pp. 679-698, 1986.
- [66] A. H. Baker, *The Windows NT Device Driver Book: A Guide for Programmers*. Upper Saddle River, NJ: Prentice Hall PTR, 1997.
- [67] G. T. Chavez and R. R. Murphy, "Exception handling for sensor fusion," in *Proceedings of the SPIE - Signal Processing, Sensor Fusion, and Target Recognition VI*, vol. 2059, pp. 142-153, SPIE, 1993.
- [68] R. R. Murphy and D. Hershberger, "Classifying and recovering from sensing failures in autonomous mobile robots," in *Proceedings of the Thirteenth National Conference on Artificial Intelligence: AAAI-96*, vol. 2, (Portland, Oregon), pp. 922-929, AAAI/IAAI, Aug. 1996.
- [69] R. P. Lippmann, "An introduction to computing with neural nets," *IEEE ASSP Magazine*, pp. 4-22, 1987.
- [70] S.-R. Lay and J.-N. Hwang, "Robust construction of radial basis function networks for classification," in *Proceedings of the IEEE International Conference on Neural Networks*, (San Francisco, CA), pp. 2037-2044, IEEE, Mar. 28-Apr. 1, 1993.
- [71] Y.-F. Wong, "How gaussian radial basis functions work," in *Proceedings of the IEEE International Joint Conference on Neural Networks*, pp. 302-309, IEEE, 1991.
- [72] D. R. Hush and B. G. Horne, "Progress in supervised neural networks," *IEEE Signal Processing Magazine*, pp. 8-39, Jan. 1993.
- [73] J. E. Shigley and C. R. Mischke, *Mechanical Engineering Design*. New York: McGraw-Hill, 5th ed., 1989.

- [74] G. E. Dieter, *Engineering Design: A Materials and Processing Approach*. New York: McGraw-Hill, 2nd ed., 1991.
- [75] Fisheries and Oceans — Canada. Inspection Services, *Metal Can Defects: Identification and Classification*, Jan. 1994.
- [76] M.-F. Lee, C. W. de Silva, E. A. Croft, and H. J. Park, "Automated screening of metal can defects using machine vision," in *Proceedings of the Second International Symposium on Intelligent Automation and Control*, (Anchorage, Alaska), pp. 175.1–175.6, WAC, May 9-14 1998.
- [77] S. Kurnianto, "Design, development, and integration of an automated herring roe grading system," Master's thesis, Department of Mechanical Engineering, University of British Columbia, Vancouver, B.C. V6T 1Z4, June 1997.
- [78] J. Budenske and M. Gini, "Sensor explication: Knowledge-based robotic plan execution through logical objects," *IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics*, vol. 27, pp. 611–625, Aug. 1997.

Appendix A

Object Model Class

A.1 Introduction

This chapter categorizes and describes the object model classes which are used to represent objects within ELSA.

A.2 Class Summary

This section briefly summarizes the object model classes. For each derived class, the inheritance tree is provided in the corresponding section

CNode

Base class for nodes in the object model structure.

CObjectNode

Derived class which represents object nodes.

CClassificationNode

Derived class which represents classification nodes.

CObjectProperties

Class derived from **CElement** to allow list representation of object properties.

CPhysicalProperties

Derived class for physical object properties.

CRelationalProperties

Derived class for relational object properties.

A.3 The Classes

class CNode

A **CNode** object represents a generic node of the object model hierarchy. It provides basic functionality: a name and links to child node(s). To allow an arbitrary number of child nodes, links are maintained in a **CList** structure. **CNode** serves as a base class for derivation of more specialized node types.

Construction/Destruction — Public Members

CNode	Constructs a CNode object.
~CNode	Destroys a CNode object.

Attributes — Public Members

GetName	Returns the name of the node.
GetNumChildren	Returns the number of child nodes.

Operations — Public Members

AddChild	Adds a pointer to a child node.
DeleteChild	Removes a child node.

Member Functions

CNode::CNode

CNode(char * strName = NULL);

strName Name of the node.

Constructs a **CNode** object.

CNode::~CNode**virtual ~CNode();**

Destroys a **CNode** object.

CNode::GetName**char * GetName() const;**

Returns the name of the **CNode** object.

CNode::GetNumChildren**int GetNumChildren() const;**

Returns the number of children this **CNode** object is the parent for.

CNode::AddChild**virtual AddChild(CNode * pNode);**

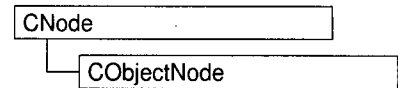
Adds a child node to the **CNode** object. This function is declared as a pure virtual function. It must be redefined by derived classes.

CNode::DeleteChild**virtual DeleteChild(CNode * pNode);**

Removes the pointer to the specified child node from the **CNode** object. This function is declared as a pure virtual function. It must be redefined by derived classes.

class CObjectNode : public CNode

A **CObjectNode** object represents a specialization of the a **CNode** object. **CObjectNodes** are used to represent the objects and features which comprise the feature layer of the object model.



Construction/Destruction — Public Members

CObjectNode	Constructs a CObjectNode object.
~CObjectNode	Destroys a CObjectNode object.

Attributes — Public Members

IsFree	Returns nonzero if the node is marked by a free tag.
GetObjectType	Returns the type of object represented by node.
GetProperties	Returns a pointer to the list of object properties.

Operations — Public Members

AddChild	Adds a pointer to a child node.
DeleteChild	Removes a child node.
AddProperty	Adds a physical or relational property to node.
DeleteProperty	Removes a physical or relational property from node.

Member Functions

CObjectNode::CObjectNode

CObjectNode(char * strName, ObjectType eType, bool bFree = FALSE);

strName Name of the object node.

eType The type of object that this node represents.

bFree Nonzero if the object should be marked by a free node tag.

Constructs a **CObjectNode** object. **eType** is an enumeration which represents the object types that are recognized by the system. These types are outlined in Table A.1. Additional types may be added by expanding the enumeration.

Table A.1: Enumeration of object types.

Value	Meaning
0	Generic
1	Linear dimension
2	Position
3	Distance
4	Line
5	Corner
6	Boundary curve
7	Boundary feature
8	Area
9	Conic
10	Regular polygon
11	Irregular polygon
12	Character
13	Volume
14	Colour
15	3D surface
16	Regular polyhedron
17	Irregular polyhedron
18	Surface feature
19	Texture
20	Force
21	Pressure
22	Sound
23	Odour
24	Mass
25	Speed
26	Temperature
27-255	Reserved for future expansion

CObjectNode::~~CObjectNode

```
virtual ~CObjectNode( );
```

Destroys a **CObjectNode** object.

CObjectNode::IsFree()

```
bool IsFree( );
```

Returns **TRUE** if this **CObjectNode** represents a free node of the object model.

CObjectNode::GetObjectType

```
ObjectType GetObjectType( ) const;
```

Returns an enumerated value representing the type of object that this **CObjectNode** represents. The types recognized by the system are outlined in Table A.1.

CObjectNode::GetObjectProperties

```
ObjectType GetObjectProperties( ) const;
```

Returns a pointer to a **CList** containing **CPhysicalProperties** and **CRelationalProperties** for the **CObjectNode**

CObjectNode::AddChild

```
AddChild(CObjectNode * pNode);
```

pNode Pointer to the child node to be added.

Adds a pointer to a child node from the **CObjectNode** object.

CObjectNode::DeleteChild

```
DeleteChild(CObjectNode * pNode);
```

pNode Pointer to the child node to be removed.

Removes the pointer to the specified child node from the **CObjectNode** object.

CObjectNode::AddProperty

AddProperty(CObjectProperty * pProperty);

pProperty Pointer to the property to be added.

Adds a physical or relational property to the list of object properties maintained by the **CObjectNode**.

CObjectNode::DeleteProperty

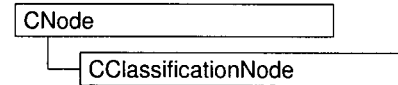
DeleteProperty(CObjectProperty * pProperty);

pProperty Pointer to the property to be removed.

Deletes a physical or relational property to the from of object properties maintained by the **CObjectNode**.

class CClassificationNode : public CNode

A **CClassificationNode** object represents a specialization of the a **CNode** object. **CClassificationNodes** are used to represent the classifications of an object. These correspond to the classification layer of the object model.



Data Members — Public Members

CFuzzyLink	Represents a pointer to a CObjectNode with a corresponding fuzzy descriptor of type CFuzzyVariable .*
-------------------	---

Construction/Destruction — Public Members

CClassificationNode	Constructs a CClassificationNode object.
~CClassificationNode	Destroys a CClassificationNode object.

Operations — Public Members

AddChild	Adds a pointer to a child node representing a primary feature for classification.
DeleteChild	Removes a child node.

Member Functions

CClassificationNode::CClassificationNode

CClassificationNode(char * strName);

strName Name of the classification node.

Creates a **CClassificationNode** object.

*The **CFuzzyVariable** class is described in Appendix C

CClassificationNode::~~CClassificationNode

virtual ~CClassificationNode();

Destroys a **CClassificationNode** object.

CClassificationNode::AddChild

AddChild(CObjectNode * pNode, CFuzzyValue fvDescriptor);

pNode Pointer to object node to add as child.

fvDescriptor Fuzzy descriptor which defines how the feature represented by the object node relates to the classification.

Adds a pointer to a child node from the **CClassificationNode** object. This pointer and the corresponding fuzzy descriptor are maintained within a list of **CFuzzyLink** objects.

CClassificationNode::DeleteChild

DeleteChild(CObjectNode * pNode);

pNode Pointer to the object node to remove.

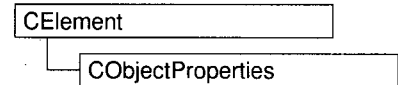
Removes the pointer to the specified child node from the **CClassificationNode** object.

Data Members**CClassificationNode::CFuzzyLink**

This structure represents the combination of a pointer to a **CObjectNode** with a corresponding fuzzy descriptor of type **CFuzzyValue**.

class **CObjectProperties** : public **CElement**

A **CObjectProperties** object represents properties of an object in a generic manner. This class is derived from **CElement** to enable object properties to be included in a generic list. **CObjectProperties** serves as a base class for derivation of more specialized property types.



Construction/Destruction — Public Members

CObjectProperties Constructs a **CObjectProperties** object.

~CObjectProperties Destroys a **CObjectProperties** object.

Member Functions

CObjectProperties::CObjectProperties

CObjectProperties();

CObjectProperties(char * strName);

strName Name of the property element.

Creates a **CObjectProperties** object.

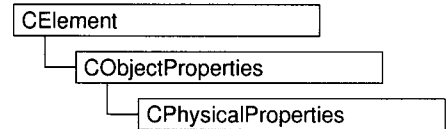
CObjectProperties::~~CObjectProperties

virtual ~CObjectProperties();

Destroys a **CObjectProperties** object.

class CPhysicalProperties : public CObjectProperties

A **CPhysicalProperties** object represents a specialization of the **CObjectProperties** object. **CPhysicalProperties** objects are used to represent physical properties of an object. These include shape, mass, position, colour, etc.



Construction/Destruction — Public Members

CPhysicalProperties	Constructs a CPhysicalProperties object.
~CPhysicalProperties	Destroys a CPhysicalProperties object.

Attributes — Public Members

GetValue	If applicable, returns a value representing the property.
GetType	Returns the type of property being represented.

Operations — Public Members

SetValue	Sets the value of the property.
SetType	Sets the type of property being represented.

Member Functions

CPhysicalProperties::CPhysicalProperties

CPhysicalProperties();

CPhysicalProperties(char * strName);

strName Name of the property element.

Creates a **CPhysicalProperties** object.

CPhysicalProperties::~CPhysicalProperties

```
virtual ~CPhysicalProperties( );
```

Destroys a **CPhysicalProperties** object.

CPhysicalProperties::GetValue

```
int GetValue( ) const;
```

```
double GetValue( );
```

Returns the value associated with the physical property, if one exists.

CPhysicalProperties::GetType

```
PropertyType GetType( ) const;
```

Returns the type of physical property represented as an enumeration. These types are outlined in Table A.2. Additional types may be added by expanding the enumeration.

CPhysicalProperties::SetValue

```
SetValue(int iVal);
```

```
SetValue(double dVal);
```

Sets the value associated with the physical property, if one exists.

CPhysicalProperties::SetType

```
SetType(PropertyType eType);
```

eType The type of property that is represented by **CPhysicalProperties** object.

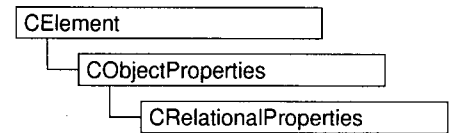
Sets the type of physical property that is represented by the **CPhysicalProperties** object. Available types are listed in Table A.2

Table A.2: Enumeration of property types.

Value	Meaning
0	Generic
1	Length
2	Width
3	Height
4	Position
5	Angle
6	Area
7	Volume
8	Colour
9	Mass
10	Texture
11	Temperature
12	Symmetry
13	Adjacency
14	Relative position
15	Relative orientation
16	Tangent
17–255	Reserved for future expansion

class CRelationalProperties : public CObjectProperties

A **CRelationalProperties** object represents a specialization of the **CObjectProperties** object. **CRelationalProperties** objects are used to represent properties of an object in relation to another. These include symmetry, adjacency, relative position, etc.



Construction/Destruction — Public Members

CRelationalProperties	Constructs a CRelationalProperties object.
~CRelationalProperties	Destroys a CRelationalProperties object.

Attributes — Public Members

To	Returns the object that is related to the current object by the relational properties.
GetValue	If applicable, returns a value representing the property.
GetType	Returns the type of property being represented.

Operations — Public Members

SetRelation	Declares the object that is related to the current object by the relational properties.
SetValue	Sets the value of the property.
SetType	Sets the type of property being represented.

Member Functions

CRelationalProperties::CRelationalProperties

CRelationalProperties();

CRelationalProperties(char * strName);

strName Name of the property element.

Creates a **CRelationalProperties** object.

CRelationalProperties::~~CRelationalProperties

virtual ~CRelationalProperties();

Destroys a **CRelationalProperties** object.

CRelationalProperties::To

CObjectNode * To() const;

Returns the object that is related to the current object by the relational properties.

CRelationalProperties::GetValue

int GetValue() const;

double GetValue();

Returns the value associated with the relational property, if one exists.

CRelationalProperties::GetType

PropertyType GetType() const;

Returns the type of relational property represented as an enumeration. These types are outlined in Table A.2. Additional types may be added by expanding the enumeration.

CRelationalProperties::SetRelation**SetRelation(CObjectNode * pObj);**

pObject Address of the object that this property is relative to.

Defines the object that is related to the current object by the relational properties.

CRelationalProperties::SetValue**SetValue(int iVal);****SetValue(double dVal);**

Sets the value associated with the relational property, if one exists.

CRelationalProperties::SetType**SetType(PropertyType eType);**

eType The type of property that is represented by **CRelationalProperties** object.

Sets the type of relational property that is represented by the **CRelationalProperties** object.

Available types are listed in Table A.2

Appendix B

Extended Logical Sensor Class

B.1 Introduction

This chapter outlines and describes the class that serves as the basis for all Extended Logical Sensor (ELS) implementations. This is an abstract base class; derived classes must provide much of the functionality of the ELS. The purpose of this class then is to provide a common structure from which logical sensors are implemented. This ensures that ELSs can interact with one another.

B.2 The Class

class CELSBase

A **CELSBase** object is a base class. From this object, specialized ELS objects may be derived. The derived classes must then provide the complete implementation of the ELS. Thus, this class serves as a template for the construction of ELS objects for specific purposes.

Construction/Destruction — Public Members

CELSBase	Constructs a CELSBase object.
~CELSBase	Destroys a CELSBase object.

Attributes — Protected Members

GetName	Returns name of ELS.
GetCOV	Returns the format of the COV.
GetFunction	Returns a description of ELS functionality.
GetDeps	Returns ELS dependencies.

Operations — Public Members

ReadData	Reads sensor data from external sources.
SendData	Sends sensor data to external sources.
ReadCommand	Reads commands from external sources.
SendCommand	Sends commands to external sources.

Operations — Protected Members

SetCOV	Sets the format of the COV.
BuildDeps	Determines dependencies on other ELSs.
Initialize	Initializes the ELS.
Calibrate	Performs calibration of the ELS.
Poll	Provides a response to queries.
Sense	Prompts ELS to provide output.
Reset	Resets all ELS parameters to initial values.
Test	Calls tests embedded in the ELS.
Select	Requests that an alternate program be chosen, if available.
Monitor	Validates ELS data.

Member Functions

CELSBase::CELSBase

CELSBase(char * strName);

strName Name of the ELS class.

Constructs a **CELSBase** object.

CELSBase::~~CELSBase

virtual ~CELSBase();

Destroys a **CELSBase** object.

CELSBase::GetName

char * GetName();

Returns the name of the ELS.

CELSBase::GetCOV

COV * GetCOV();

Returns the Characteristic Output Vector (COV) for the ELS. The COV is a vector of types. It maintained as a private data member. Supported types are listed in Table B.1.

CELSBase::GetFunction

char * GetFunction();

Returns a description of the functionality that this sensor provides. The description is in human readable form. This is intended to provide a mechanism by which a user may browse through an ELS library and determine the capabilities and/or suitability of a particular ELS for an application.

Table B.1: Abstract and derived types.

Keyword	Data type	Bits and range
ANY	Generic data type	Unspecified
ARRAY	Array	Sequence of elements, indexed from 0
BOOL	Bit or Boolean	TRUE or FALSE
BYTE	Byte	Bit string of 8 bits
DOUBLE	Double	Real value represented by 64 bits
ENUM	Enumeration	Value of type restricted to enumeration
FLOAT	Float	Real value represented by 32 bits
INT	Integer	-32768 through +32767
LONG DOUBLE	Long Double	Real value represented by 128 bits
OBJECTREF	Handle for an object	Unspecified
STRING	Text string	An array of characters
TIME	Duration	Implementation dependent
TIMESTAMP	UTC	UTC
UINT	Unsigned Integer	0 through 65534
UNICODE	Unicode	An array of characters 1-2 bytes per character
USINT	Unsigned Short Integer	0 through 255

CELSBase::GetDeps

```
DepList * GetDeps( );
```

Returns a list of each sensor that provides input to the programs contained in the ELS.

CELSBase::ReadData

```
ReadData(COV * data);
```

data Input sensor data in form of COV.

This the first of four functions that comprise the public interface of an **CELSBase** object. It serves to extract data from various input sources — transducers and logical sensors. Data is expected in the form of a COV. For transducers, this function must be overloaded to accept the output of the transducer.

CELSBase::SendData

```
COV * SendData( );
```

The second ELS public interface function, **SendData** sends sensor data generated by the ELS to external sources. Data is in the form of the COV for the ELS.

CELSBase::ReadCommand

```
ReadCommand(CommandID eCommand);
```

eCommand Input command.

The third ELS public interface function. It serves to extract commands that may be sent from other ELSs or other system modules. These commands are of an enumerated type **CommandID**, as listed in Table B.2.

Table B.2: Enumeration of ELS control commands: CommandID.

Value	Meaning
0	Initialize
1	Calibrate
2	Poll:Name
3	Poll:COV
4	Poll:Function
5	Poll:Deps
6	Sense
7	Reset
8	Test
9	Select
10	Monitor
11–255	Reserved for future expansion

CELSBase::SendCommand

```
CommandID eCommand SendCommand( );
```

The fourth ELS public interface function, **SendCommand** sends commands of the enumerated type **CommandID** to external sources.

CELSBase::SetCOV

SetCOV(COV * pCOV);

pCOV Address of COV.

This function is used to define the format of the COV.

CELSBase::BuildDeps

virtual BuildDeps();

Determines dependencies on other logical sensors by identifying sources of input for the programs utilized within the ELS.

CELSBase::Initialize

virtual Initialize();

Initializes an ELS upon creation.

CELSBase::Calibrate

virtual Calibrate();

Performs calibration of the logical sensor using built-in private member functions.

CELSBase::Poll

char * Poll(CommandID eCommand);

COV * Poll(CommandID eCommand);

DepList * Poll(CommandID eCommand);

eCommand Polling command.

Responds to polling requests by obtaining and returning the queried information. This function is overloaded to enable the return of each type of data maintained within the Logical Sensor Characteristics.

CELSBase::Sense

virtual Sense();

Prompts the ELS to provide output. This command may be used for temporal synchronization of sensing events.

CELSBase::Reset

virtual Reset();

Resets, to their original values, all logical sensor parameters which may have been modified.

CELSBase::Test

virtual Test();

Tests the functionality of the ELS by invoking one or more embedded tests.

CELSBase::Select

virtual Select();

Prompts the ELS to select an alternate program, if one is available. There are no arguments to the function because the user does not have the ability to select the most appropriate replacement. Providing knowledge of the program(s) operation to enable direct user intervention would compromise the encapsulation.

CELSBase::Monitor

virtual Monitor();

Used to validate the data obtained by the ELS. Typically, monitoring simply involves checking the data value against an expected range. If the data is in range, it is accepted; if not, an exception is thrown.

Appendix C

Fuzzy Variable Class

C.1 Introduction

This chapter categorizes and describes the fuzzy variable classes. These are general purpose classes, written as class templates, to ensure applicability to many applications. Within ELSA, these classes are used to represent the fuzzy links of the object model and the Rulebase for the Inference Engine.

These classes include:

- Classes for representing fuzzy values and degree of membership.
- Classes for representing fuzzy sets.

C.2 Class Summary

This section briefly summarizes the fuzzy variable classes. For each derived class, the inheritance tree is provided in the corresponding section.

CFuzzyDegree

Represents the degree of membership of a value within a fuzzy set.

CFuzzyVariable

Provides a representation of fuzzy variables and linguistic variables.

CFuzzyMember

Base class for fuzzy sets (membership functions).

CFuzzyGeomMember

Derived class which represents features common to geometrical membership functions. This class is intended to be used as a base for further derivation.

CFuzzyTriMember

Represents fuzzy membership functions which are triangular in shape.

CFuzzyTrapMember

Represents fuzzy membership functions which are of a trapezoidal shape.

CFuzzyArrayMember

Represents fuzzy membership functions which are constructed from an array of points.

C.3 The Classes

template<class C> class CFuzzyDegree

A **CFuzzyDegree** object represents the degree to which an input value is a member of a given fuzzy set. For example, consider membership in the set **HIGH**, and a variable **temp**. The degree to which **temp** is a member of the set **HIGH** is represented by a **CFuzzyDegree** object. The degree of membership is expressed as a value in the range [0–1].

Construction/Destruction — Public Members

CFuzzyDegree	Constructs a CFuzzyDegree object.
~CFuzzyDegree	Destroys a CFuzzyDegree object.

Operators

operator &&	Performs the intersection of two CFuzzyDegree values.
operator 	Performs the union of two CFuzzyDegree values.
operator !	Negates a CFuzzyDegree value.

Member Functions

CFuzzyDegree::CFuzzyDegree

CFuzzyDegree();

CFuzzyDegree(C cDeg);

cDeg Degree of membership in range [0–1]. For integer values, the degree of membership is expressed as a scaled integer in range [0–255]; otherwise, expressed as a decimal number, e.g. 0.56.

Constructs a **CFuzzyDegree** object. Currently, **int** and **float** values are supported.

CFuzzyDegree::~~CFuzzyDegree

virtual ~CFuzzyDegree();

Destroys a **CFuzzyDegree** object.

CFuzzyDegree::operator &&

CFuzzyDegree CFuzzyDegree::operator && (CFuzzyDegree &);

Allows **CFuzzyDegree** objects to be combined through a union or **AND** operation. The minimum of the two **CFuzzyDegree** values is returned.

CFuzzyDegree::operator ||

CFuzzyDegree CFuzzyDegree::operator || (CFuzzyDegree &);

Allows **CFuzzyDegree** objects to be combined through an intersection or **OR** operation. The maximum of the two **CFuzzyDegree** values is returned.

CFuzzyDegree::operator !

CFuzzyDegree CFuzzyDegree::operator ! ();

Negates a **CFuzzyDegree** object. This is a **NOT** operation.

template<class C> class CFuzzyVariable

A **CFuzzyVariable** object represents a fuzzy variable and its corresponding linguistic variable. For example, the linguistic variable **Temperature** might be programmed as follows:

```
CFuzzyVariable<int> temp("Temperature", 0, 100);
enum {COLD, WARM, HOT};
temp.AddValueSet(COLD, new CFuzzyTrapMember("cold", 0, 0, 10, 20, 1));
temp.AddValueSet(WARM, new CFuzzyTrapMember("warm", 10, 20, 25, 35, 1));
temp.AddValueSet(HOT, new CFuzzyTrapMember("hot", 25, 35, 100, 100, 1));
```

In this case, trapezoidal membership functions have been used, but any other kind of function which has been implemented may also be used.

Construction/Destruction — Public Members

CFuzzyVariable	Constructs a CFuzzyVariable object.
----------------	--

Attributes — Public Members

GetName	If named, returns the name of the value.
GetValue	Returns the non-fuzzy value.
GetFuzzyValue	Returns the fuzzy value (fuzzy set).
GetMin	Returns the lower extent of the variable's universe of discourse.
GetMax	Returns the upper extent of the variable's universe of discourse.
GetValueSet	Returns the fuzzy set at the given position.
Is	Returns the membership value of the fuzzy set at the given array position.

Operations — Public Members

SetValue	Sets the non-fuzzy variable value and returns the old.
SetFuzzyValue	Sets the fuzzy variable value (fuzzy set) and returns the old value.

AddValueSet Adds the fuzzy set to the array of affiliated sets.

Operators

operator C Allow extracting of fuzzy variable through casting or its use in some clear context.

operator = Allow direct assignment to fuzzy variable.

Member Functions

CFuzzyVariable::CFuzzyVariable

CFuzzyVariable(C min = 0, C max = 1)

CFuzzyVariable(const * char &strName, C min = 0, C max = 1)

strName Name of fuzzy variable.

min Lower extent of universe of discourse for variable.

max Upper extent of universe of discourse for variable.

Constructs a **CFuzzyVariable** object. The universe of discourse for the fuzzy variable (and associated fuzzy sets) is defined by the range [**min-max**].

CFuzzyVariable::GetName

char * GetName() const;

Returns the name of the **CFuzzyVariable** object.

CFuzzyVariable::GetValue

C GetValue() const;

Returns the non-fuzzy value of the **CFuzzyVariable** object.

CFuzzyVariable::GetFuzzyValue

CFuzzyMember * GetFuzzyValue() const;

Returns the fuzzy value (fuzzy set) of the **CFuzzyVariable** object.

CFuzzyVariable::GetMin

C GetMin() const;

Returns the lower extent (minimum) of the universe of discourse for the **CFuzzyVariable** object.

CFuzzyVariable::GetMax

C GetMax() const;

Returns the upper extent (maximum) of the universe of discourse for the **CFuzzyVariable** object.

CFuzzyVariable::GetValueSet

CFuzzyMember * GetValueSet(int i) const;

i Index to array position.

Returns the fuzzy set at array position *i*.

CFuzzyVariable::Is

double Is(int i);

i Index to array position.

Gets the membership value of the fuzzy set at array position *i*.

CFuzzyVariable::SetValue

```
C SetValue(C cVal);
```

cVal The value to set the **CFuzzyVariable** object to.

Sets the non-fuzzy value of the **CFuzzyVariable** object to **cVal** and returns the old value.

CFuzzyVariable::SetFuzzyValue

```
CFuzzyMember * SetFuzzyValue(CFuzzyMember * fmVal);
```

fmVal The fuzzy value to set the **CFuzzyVariable** object to.

Sets the fuzzy value of the **CFuzzyVariable** object to **cVal** and returns the old value.

CFuzzyVariable::AddValueSet

```
AddValueSet(CFuzzyMember & fmSet);
```

```
AddValueSet(CFuzzyMember * fmSet);
```

```
AddValueSet(int i, CFuzzyMember & fmSet);
```

```
AddValueSet(int i, CFuzzyMember & fmSet);
```

i Index to where new set will be added.

fmSet Address of the set to add to the **CFuzzyVariable**.

Adds the fuzzy set **fmSet** to the array of affiliated sets. If **i** is specified, the set is added at that location, overwriting the set that was there. In the case that **i** is unspecified, the set is added to the end, extending the array.

CFuzzyVariable::operator C

```
template<class C> CFuzzyVariable<C>::operator C ( );
```

Allows a **CFuzzyVariable** object to be casted.

CFuzzyVariable::operator =

```
template<class C> C & CFuzzyVariable<C>::operator = (const CFuzzyMember<C> &)  
const;
```

Allows direct assignment of a **CFuzzyVariable** object.

template<class C> class CFuzzyMember

A **CFuzzyMember** object serves as a base class for the representation of a fuzzy set.

Construction/Destruction — Public Members

CFuzzyMember Constructs a **CFuzzyMember** object.

~CFuzzyMember Destroys a **CFuzzyMember** object.

Attributes — Public Members

GetName Returns the name of the fuzzy set.

Is Returns the degree of membership.

Member Functions

CFuzzyMember::CFuzzyMember

CFuzzyMember(const C *xmin*, const C *xmax*);

CFuzzyMember(char * *strName*, const C *xmin*, const C *xmax*);

strName Name of the set.

xmin x coordinate for lower extent of set base.

xmax x coordinate for upper extent of set base.

Constructs a **CFuzzyMember** object. The variables **xmin** and **xmax** define the base of the fuzzy set. Derived classes provide the representation for height and shape.

CFuzzyMember::~~CFuzzyMember

virtual ~CFuzzyMember();

Destroys **CFuzzyMember** object. Deallocates space occupied by **strName**.

CFuzzyMember::GetName

```
char * GetName( ) const;
```

Returns the name of the the **CFuzzyMember** object.

CFuzzyMember::Is

```
virtual CFuzzyDegree Is(const C cValue) const;
```

cValue Input value for which to determine membership.

Returns the degree of membership of **cValue** within the **CFuzzyMember** object. This function is declared as a pure virtual function. The function must be redefined by derived classes.

```
template<class C>  
class CFuzzyGeomMember : public CFuzzyMember<C>
```

A **CFuzzyGeomMember** object serves as a base class for the representation of a fuzzy value within a fuzzy set.

CFuzzyMember

CFuzzyGeomMember

Construction/Destruction — Public Members

CFuzzyGeomMember Constructs a **CFuzzyGeomMember** object.

~CFuzzyGeomMember Destroys a **CFuzzyGeomMember** object.

Attributes — Public Members

Is Returns the degree of membership.

GetArea Area of membership function.

GetCentroid Centroid of membership function.

GetMoment Moment of area of membership function.

Scale Scales the membership function to a given degree.

Clip Clips the membership function at a given degree.

Member Functions

```
CFuzzyGeomMember::CFuzzyGeomMember
```

```
CGeomFuzzyMember(const C xmin, const C Xmax);
```

```
CGeomFuzzyMember(char * strName, const C Xmin, const C xmax);
```

strName Name of the set.

xmin x coordinate for lower extent of set base.

xmax x coordinate for upper extent of set base.

Constructs a **CGeomFuzzyMember** object. On construction, the area, centroid, and moment of area are computed. The variables **xmin** and **xmax** define the base of the fuzzy set. Derived classes provide the representation for height and shape.

CFuzzyGeomMember::~CFuzzyGeomMember

~CFuzzyGeomMember();

Destroys a **CFuzzyMember** object.

CFuzzyGeomMember::Is

virtual CFuzzyDegree Is(const C cValue) const;

cValue Input value for which to determine membership.

Returns the degree of membership of **cValue** within the **CFuzzyGeomMember** object. This function is declared as a pure virtual function. The function must be redefined by derived classes.

CFuzzyGeomMember::GetArea

virtual C GetArea() const;

Returns a value which represents the area of the membership function. This function is declared as a pure virtual function. The function must be redefined by derived classes.

CFuzzyGeomMember::GetCentroid

virtual C GetCentroid() const;

Returns a value which represents the centroid of the membership function. This function is declared as a pure virtual function. The function must be redefined by derived classes.

CFuzzyGeomMember::GetMoment

virtual C GetMoment() const;

Returns a value which represents the moment of area for the membership function. This function is declared as a pure virtual function. The function must be redefined by derived classes.

CFuzzyGeomMember::Scale

```
virtual CFuzzyGeomMember<C> * Scale(const CFuzzyDegree fdDegree) const;
```

fdDegree Degree at which to scale membership function.

This function scales the current membership function to a given degree. Returns the new scaled membership function. This function is declared as a pure virtual function. The function must be redefined by derived classes.

CFuzzyGeomMember::Clip

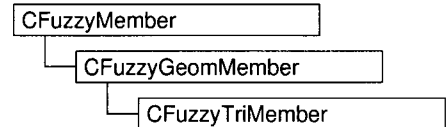
```
virtual CFuzzyGeomMember<C> * Clip(const CFuzzyDegree fdDegree) const;
```

fdDegree Degree at which to clip membership function.

This function clips the current membership function to a given degree. Returns the new clipped membership function. This function is declared as a pure virtual function. The function must be redefined by derived classes.

```
template<class C>
class CFuzzyTriMember : public CFuzzyGeomMember<C>
```

A **CFuzzyTriMember** object implements a triangular fuzzy set.



Construction/Destruction — Public Members

CFuzzyTriMember	Constructs a CFuzzyTriMember object.
~CFuzzyTriMember	Destroys a CFuzzyTriMember object.

Attributes — Public Members

Is	Returns the degree of membership.
GetArea	Area of membership function.
GetCentroid	Centroid of membership function.
GetMoment	Moment of area of membership function.
Scale	Scales the membership function to a given degree.
Clip	Clips the membership function at a given degree.

Member Functions

```
CFuzzyTriMember::CFuzzyTriMember
```

```
CFuzzyTriMember(C x1, C x1, C x1, C h);
```

```
CFuzzyTriMember(char * strName, C x1, C x1, C x1, C h);
```

strName Name of the fuzzy set.

x1 x coordinate of left side (lower extent) of triangle base.

x2 x coordinate of peak of triangle.

$x3$ x coordinate of right side (upper extent) of triangle base.

h Height of triangle. This is typically 1.0.

This class provides a template for the creation of fuzzy membership functions with a triangular shape. The corner coordinates and height of the function must be specified. These are illustrated in Figure C.1

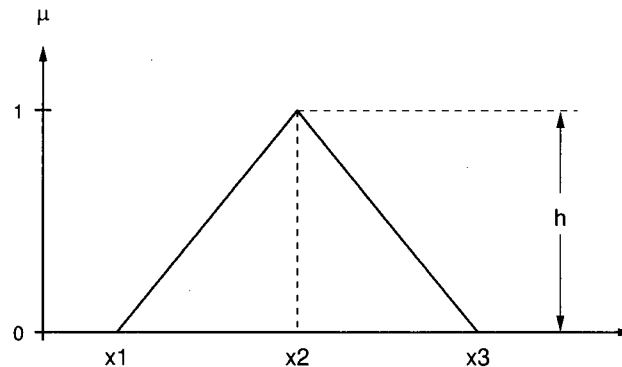


Figure C.1: Triangular membership function.

The area, centroid, and moment of area for the triangle are computed on initialization. Currently, **int**, **float**, and **double** types are supported by the template. For other data types, the appropriate member functions must be written.

CFuzzyTriMember::~CFuzzyTriMember

~CFuzzyTriMember();

Destroys a **CFuzzyTriMember** object.

CFuzzyTriMember::Is

CFuzzyDegree Is(const C cValue) const;

cValue Input value for which to determine membership.

Returns the degree of membership of **cValue** within the **CFuzzyTriMember** object.

CFuzzyTriMember::GetArea

C GetArea() const;

Returns a value which represents the area of the membership function.

CFuzzyTriMember::GetCentroid

C GetCentroid() const;

Returns a value which represents the centroid of the membership function.

CFuzzyTriMember::GetMoment

C GetMoment() const;

Returns a value which represents the moment of area for the membership function.

CFuzzyTriMember::Scale

CFuzzyGeomMember<C> * Scale(const CFuzzyDegree fdDegree) const;

fdDegree Degree at which to scale membership function.

This function scales the current membership function to a given degree. Returns the new scaled membership function.

CFuzzyTriMember::Clip

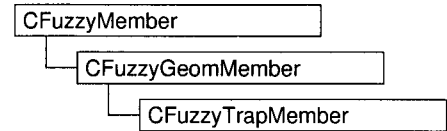
CFuzzyGeomMember<C> * Clip(const CFuzzyDegree fdDegree) const;

fdDegree Degree at which to clip membership function.

This function clips the current membership function to a given degree. Returns the new clipped membership function.

```
template<class C>
class CFuzzyTrapMember : public CFuzzyGeomMember<C>
```

A **CFuzzyTrapMember** object implements a trapezoidal fuzzy set.



Construction/Destruction — Public Members

CFuzzyTrapMember	Constructs a CFuzzyTrapMember object.
~CFuzzyTrapMember	Destroys a CFuzzyTrapMember object.

Attributes — Public Members

Is	Returns the degree of membership.
GetArea	Area of membership function.
GetCentroid	Centroid of membership function.
GetMoment	Moment of area of membership function.
Scale	Scales the membership function to a given degree.
Clip	Clips the membership function at a given degree.

Member Functions

```
CFuzzyTrapMember::CFuzzyTrapMember
```

```
CFuzzyTrapMember(C x1, C x2, C x3, C x4, C h);
```

```
CFuzzyTrapMember(char * strName, C x1, C x2, C x3, C x4, C h);
```

strName Name of the fuzzy set.

x1 x coordinate of left side of trapezoid base.

x2 x coordinate of left side of trapezoid 'plateau.'

x_4 x coordinate of right side of trapezoid 'plateau.'

x_4 x coordinate of right side of trapezoid base.

h Height of trapezoid. This is typically 1.0.

This class provides a template for the creation of fuzzy membership functions with a trapezoid shape. The corner coordinates and height of the function must be specified. These are illustrated in Figure C.2

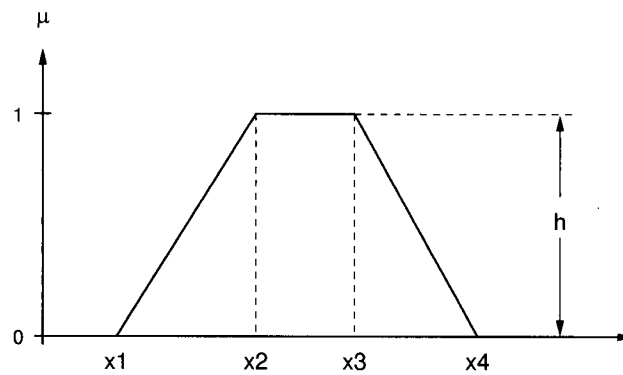


Figure C.2: Trapezoidal membership function.

The area, centroid, and moment of area for the trapezoid are computed on initialization. Currently, **int**, **float**, and **double** types are supported by the template. For other data types, the appropriate member functions must be written.

CFuzzyTrapMember::~CFuzzyTrapMember

~CFuzzyTrapMember();

Destroys a **CFuzzyTrapMember** object.

CFuzzyTrapMember::Is

CFuzzyDegree Is(const C cValue) const;

cValue Input value for which to determine membership.

Returns the degree of membership of **value** within the **CFuzzyTrapMember** object.

CFuzzyTrapMember::GetArea

C GetArea() const;

Returns a value which represents the area of the membership function.

CFuzzyTrapMember::GetCentroid

C GetCentroid() const;

Returns a value which represents the centroid of the membership function.

CFuzzyTrapMember::GetMoment

C GetMoment() const;

Returns a value which represents the moment of area for the membership function.

CFuzzyTrapMember::Scale

CFuzzyGeomMember<C> * Scale(const CFuzzyDegree fdDegree) const;

fdDegree Degree at which to scale membership function.

This function scales the current membership function to a given degree. Returns the new scaled membership function.

CFuzzyTrapMember::Clip

CFuzzyGeomMember<C> * Clip(const CFuzzyDegree fdDegree) const;

fdDegree Degree at which to clip membership function.

This function clips the current membership function to a given degree. Returns the new clipped membership function.

template<class C> class CFuzzyArrayMember : public CFuzzyMember

A **CFuzzyArrayMember** object implements a fuzzy set as an array of points. The number of elements in the array corresponds to the number of points including and between the limits of the set.

CFuzzyMember

CFuzzyArrayMember

Points are distributed evenly. Each value represents the height of the function at a given horizontal location.

Construction/Destruction — Public Members

CFuzzyArrayMember	Constructs a CFuzzyArrayMember object.
~CFuzzyArrayMember	Destroys a CFuzzyArrayMember object.

Attributes — Public Members

Is	Returns the degree of membership.
GetNumVals	Returns the size of the array.
GetVal	Get value from array.
GetMoment	Returns moment of area of the membership function.

Attributes — Public Members

PutVal	Put value into array
---------------	----------------------

Operators

operator []	Access array via [].
--------------------	----------------------

Member Functions

CFuzzyArrayMember::CFuzzyArrayMember

CFuzzyMember();

CFuzzyMember(int iSize, C xmin, C xmax);

CFuzzyMember(char * strName, int iSize, C xmin, C xmax);

strName Name of the set.

iSize Number of points used for representation.

xmin x coordinate for lower extent of set base.

xmax x coordinate for upper extent of set base.

Constructs a **CFuzzyArrayMember** object. The variables **xmin** and **xmax** define the base of the fuzzy set; **iSize** defines the resolution of the representation.

CFuzzyArrayMember::~~CFuzzyArrayMember

virtual ~CFuzzyMember();

Destroys **CFuzzyArrayMember** object. Deallocates space occupied by **strName**.

CFuzzyMember::Is

CFuzzyDegree Is(const C cValue) const;

cValue Input value for which to determine membership.

Returns the degree of membership of **cValue** within the **CFuzzyArrayMember** object.

CFuzzyMember::GetNumVals

int GetNumVals() const;

Returns the number of elements in the array.

CFuzzyMember::GetVal**C GetVal(int i) const;***i* Index to array position.

Gets a value from the array at the position specified by *i*.

CFuzzyMember::GetMoment**C GetMoment(C min, C max) const;***min* Minimum of range.*max* Maximum of range.

Returns the moment of area for the fuzzy set.

CFuzzyMember::PutVal**PutVal(int i, C cVal);***i* Index to array position.*cVal* The value to place into array.

Puts a value into the array at the position specified by *i*.

CFuzzyArrayMember::operator []**C & CFuzzyArrayMember::operator [] (int i);***i* Index to array position.

Allows the array of **CFuzzyArrayMember** objects via []. Returns the value of the array at the position specified by *i*.

Appendix D

Rulebase Classes

D.1 Introduction

This chapter categorizes and describes a number of classes used to represent fuzzy rules, rulebases, and provide an inference mechanism. Fuzzy rulebases are built up from fuzzy rules together with fuzzy inputs and outputs. Input variables may be applied to the rulebase and using the built in inference and defuzzification functions, an inference may be made.

D.2 Class Summary

This section briefly summarizes the rulebase classes.

CFuzzyClause

Defines a structure which pairs a fuzzy variable and a fuzzy value.

CFuzzyRule

Represents an antecedent/consequent rule for fuzzy inference.

CFuzzyRulebase

Builds a rulebase of **CFuzzyRule** objects from which decisions may be inferred.

D.3 The Classes

class CFuzzyClause

A **CFuzzyClause** object represents a structure containing a fuzzy variable and fuzzy value. This pair is used build antecedents and consequents for **CFuzzyRule** objects.

Data Members — Public Members

<i>pVar</i>	Pointer to linguistic variable of clause.
<i>pValue</i>	Pointer to linguistic value of clause.

Construction/Destruction — Public Members

<i>CFuzzyClause</i>	Constructs a CFuzzyClause object.
---------------------	--

Member Functions

CFuzzyClause::CFuzzyClause

CFuzzyClause();

CFuzzyClause(**CFuzzyValue** * *fvVar*, **CFuzzyMember** * *fmSet*);

CFuzzyClause(**CFuzzyValue** & *fvVar*, **CFuzzyMember** & *fmSet*);

fvVar Address of linguistic variable.

fmSet Address of linguistic value.

Constructs a **CFuzzyClause** object. Initializes a **CFuzzyVariable** and a **CFuzzyMember** during creation.

Member Functions

CFuzzyClause::pVar

Pointer to a **CFuzzyVariable** object which represents a linguistic variable.

CFuzzyClause::pValue

Pointer to a **CFuzzySet** object which represents a linguistic value.

class CFuzzyRule

A **CFuzzyRule** object represents an antecedent/consequent type rule. There may be an arbitrary number of antecedents and consequents. The following example illustrates how rules are constructed:

```
CFuzzyRule r0;
r0.AddLHS(new CFuzzyClause(Error, PositiveHigh));
r0.AddLHS(new CFuzzyClause(deltaError, Zero));
r0.AddRHS(new CFuzzyClause(Torque, NegativeHigh));
r0.Weight(0.9);
```

Here, *Error*, *deltaError*, and *Torque* are linguistic variables represented as **CFuzzyVariable** objects. *PositiveHigh*, *Zero*, and *NegativeHigh* are linguistic values represented as **CFuzzyMember** objects. This rule is equivalent to:

IF *Error* IS *positive_high* AND *deltaError* is *zero* THEN *Torque* = *negative_high*, weight = 0.9

Construction/Destruction — Public Members

CFuzzyRule	Constructs a CFuzzyRule object.
-------------------	--

Attributes — Public Members

GetNumLHS	Returns the total number of antecedent clauses.
GetNumRHS	Returns the total number of consequent clauses.
GetLHS	Returns the antecedent clause from the specified position.
GetRHS	Returns the consequent clause from the specified position.
GetWeight	Returns the rule weight.
GetAggregateValue	Returns the current value of the aggregation of the antecedent rule inputs.

Operations — Public Members

AddLHS	Adds an antecedent to the rule.
---------------	---------------------------------

AddRHS	Adds a consequent to the rule.
SetWeight	Sets the rule weight.
Aggregate	Aggregates the antecedent rule inputs and returns the result.

Member Functions

CFuzzyRule::CFuzzyRule

CFuzzyRule();

Constructs a **CFuzzyRule** object.

CFuzzyRule::GetNumLHS

int GetNumLHS() **const**;

Returns the total number of antecedent clauses for the **CFuzzyRule** object.

CFuzzyRule::GetNumRHS

int GetNumRHS() **const**;

Returns the total number of consequent clauses for the **CFuzzyRule** object.

CFuzzyRule::GetLHS

CFuzzyClause * GetLHS(int i) **const**;

i Index to array position.

Returns the antecedent clause from the position indicated by *i*.

CFuzzyRule::GetRHS

CFuzzyClause * GetRHS(int i) **const**;

i Index to array position.

Returns the consequent clause from the position indicated by *i*.

CFuzzyRule::GetWeight

double GetWeight() const;

Returns the rule weight.

CFuzzyRule::GetAggregateValue

double GetAggregateValue() const;

Returns the current value of the aggregation of the antecedent rules.

CFuzzyRule::AddLHS

AddLHS(CFuzzyClause * cfClause);

AddLHS(CFuzzyClause & cfClause);

cfClause Address of **CFuzzyClause** object to add to rule.

Adds an antecedent, in the form of a **CFuzzyClause** object, to the rule.

CFuzzyRule::AddRHS

AddRHS(CFuzzyClause * cfClause);

AddRHS(CFuzzyClause & cfClause);

cfClause **CFuzzyClause** object to add to rule.

Adds a consequent, in the form of a **CFuzzyClause** object, to the rule.

CFuzzyRule::SetWeight

double SetWeight(double dWeight);

dWeight The rule weight expressed as a value in the range [0–1].

Sets the weight to **dWeight** and returns the old weight. The default weight is 1.0.

CFuzzyRule::Aggregate

double Aggregate();

Aggregates the antecedent rule inputs and returns the result.

class CFuzzyRuleBase

A **CFuzzyRuleBase** object represents a collection of **CFuzzyRule** objects. The class provides a number of member functions which allow input and output variables to be assigned to the rulebase and inferences to be made. The inferences may also be defuzzified.

Construction/Destruction — Public Members

CFuzzyRuleBase	Constructs a CFuzzyRule object.
-----------------------	--

Attributes — Public Members

GetNumRules	Returns the total number of rules in rulebase.
--------------------	--

GetNumInputVars	Returns the total number of input variables.
------------------------	--

GetNumOutputVars	Returns the total number of output variables.
-------------------------	---

Operations — Public Members

AddRule	Adds a rule to the rulebase.
----------------	------------------------------

AddInputVar	Adds an input variable to the rulebase.
--------------------	---

AddOutputVar	Adds an output variable to the rulebase.
---------------------	--

Resolution	Returns the output fuzzy set resolution.
-------------------	--

Inference	Computes the inference for a given fuzzy variable and fuzzy value.
------------------	--

Evaluate	Computes the output set for a given variable, combining the inference results of all rules.
-----------------	---

AggregateAll	Aggregates all rules.
---------------------	-----------------------

EvaluateAll	Evaluates all rules.
--------------------	----------------------

DefuzzifyAll	Defuzzifies all output variables.
---------------------	-----------------------------------

Member Functions

CFuzzyRuleBase::CFuzzyRuleBase

CFuzzyRuleBase();

CFuzzyRuleBase(const char * strName);

strName Name of rulebase.

Constructs a **CFuzzyRuleBase** object.

CFuzzyRuleBase::GetNumRules

int GetNumRules();

Returns the total number of rules contained within the **CFuzzyRuleBase** object.

CFuzzyRuleBase::GetNumInputVars

int GetNumInputVars();

Returns the total number of input variables to the **CFuzzyRuleBase** object.

CFuzzyRuleBase::GetNumOutputVars

int GetNumOutputVars();

Returns the total number of output variables from the **CFuzzyRuleBase** object.

CFuzzyRuleBase::AddRule

AddRule(CFuzzyRule * pRule);

AddRule(CFuzzyRule & pRule);

pRule Address of rule.

Adds a rule of form **CFuzzyRule** to the rulebase.

CFuzzyRuleBase::AddInputVar

AddInputVar(CFuzzyVariable * pVar);

AddInputVar(CFuzzyVariable & pVar);

pVar Address of fuzzy variable.

Adds an input variable of the form **CFuzzyVar** to the rulebase.

CFuzzyRuleBase::AddOutputVar

AddOutputVar(CFuzzyVariable * pVar);

AddOutputVar(CFuzzyVariable & pVar);

pVar Address of fuzzy variable.

Adds an output variable of the form **CFuzzyVar** to the rulebase.

CFuzzyRuleBase::Resolution

int Resolution() const;

Returns the resolution of the output fuzzy set.

CFuzzyRuleBase::Inference

Inference(CFuzzyVariable * pVar, CFuzzyMember * pSet, double dMatch, CFuzzyArrayMember & pResult);

pVar Address of fuzzy variable.

pSet Address of fuzzy value.

dMatch Result of antecedent aggregation.

pResult Address of resultant fuzzy set.

Computes the inference set for a variable, **pVar** of type **CFuzzyVariable**, and value, **pSet** of type **CFuzzyMember**, using value **dMatch** which is the result of aggregation of the antecedents. The result, **pResult**, is stored in a **CFuzzyArrayMember** object.

CFuzzyRuleBase::Evaluate

Evaluate(CFuzzyVariable * pVar, CFuzzyArrayMember & pResult);

pVar Address of fuzzy variable.

pResult Address of resultant fuzzy set.

Computes the output set for the variable **pVar**, combining the **Inference()** results of all rules. The result, **pResult**, is stored in a **CFuzzyArrayMember** object.

CFuzzyRuleBase::AggregateAll

AggregateAll();

Computes the aggregate of all rules in the **CFuzzyRuleBase** object.

CFuzzyRuleBase::EvaluateAll

EvaluateAll();

Evaluates all variables in the **CFuzzyRuleBase** object.

CFuzzyRuleBase::DefuzzifyAll

DefuzzifyAll();

Defuzzifies all output variables in the **CFuzzyRuleBase** object.

Appendix E

Support Classes

E.1 Introduction

This chapter categorizes and describes a number of classes which are used to support the other classes described in Appendices A–D.

E.2 Class Summary

This section briefly summarizes the support classes.

Max

Returns the maximum of two supplied values.

Min

Returns the minimum of two supplied values.

CElement

A standard element to be linked into lists.

E.3 The Classes

template<class T> T Max

Max is used to compute the maximum of two values. These values may be of any defined type.

Max

T Max(T x, T y)

x First object.

y Second object.

Compares **x** and **y**. Returns whichever is larger.

template<class T> T Min

Min is used to compute the minimum of two values. These values may be of any defined type.

Min

T Min(T x, T y)

x First object.

y Second object.

Compares **x** and **y**. Returns whichever is smaller.

template<class T> class CElement

A **CElement** object represents an element of a list. It may be used in singly or doubly linked lists.

Data Members — Public Members

pSuc	Forward pointer (to successor).
pPre	Backward pointer (to predecessor).
tData	Element data.

Construction/Destruction — Public Members

CElement	Constructs a CElement object.
-----------------	--------------------------------------

Member Functions

CElement::CElement

CElement() CElement(CElement *s, CElement *p, T d)

s Successor element.

p Predecessor element.

d Element data.

Constructs a **CElement** object. Initializes data members.

Data Members

CElement::pSuc

This is a pointer to an successor **CElement**.

CElement::pPre

This is a pointer to an predecessor **CElement**.

CElement::tData

This data member represents data of type **T**, as specified by the class template.