Instrumented Sensor System Architecture

Mohamed Dekhil and Thomas C. Henderson

UUSC-97-004

Department of Computer Science
University of Utah
Salt Lake City, UT 84112 USA

March 1997

Abstract

Sensor systems are becoming ubiquitous throughout society, yet their design, construction and operation is still more of an art than a science. In this paper, we define, develop, and apply a formal semantics for sensor systems that provides a theoretical framework for an integrated software architecture for modeling sensor-based control systems. Our goal is to develop a design framework which allows the user to model, simulate and experiment with different versions of a sensor system. This includes the ability to build and modify multisensor systems and to monitor and debug both the output of the system and the affect of any modification in terms of robustness, efficiency, and error measures. The notion of *Instrumented Logical Sensor Systems* (ILSS) that are derived from a modeling and design methodology is introduced. The instrumented sensor approach is based on a sensori-computational model which defines the components of the sensor system in terms of their functionality, accuracy, robustness and efficiency. This approach provides a uniform specification language to define sensor systems as a composition of smaller, predefined components. From a software engineering standpoint, this addresses the issues of modularity, reusability, and reliability for building complex systems. The proposed framework is implemented as a component library in a visual programming system called "CWave" that is being developed at the University of Utah.

This work was supported in part by the Advanced Research Projects agency under Army Research Office grants number DAAH04-93-G-0420 and by NSF grant CDA 9024721.

1 Introduction

In any closed-loop control system, sensors are used to provide the feedback information that represents the current status of the system and the environmental uncertainties. Building a sensor system for a certain application is a process that includes the analysis of the system requirements, a model of the environment, the determination of system behavior under different conditions, and the selection of suitable sensors. The next step in building the sensor system is to assemble the hardware components and to develop the necessary software modules for data fusion and interpretation. Finally, the system is tested and the performance is analyzed. Once the system is built, it is difficult to monitor the different components of the system for the purpose of testing, debugging and analysis. It is also hard to evaluate the system in terms of time complexity, space complexity, robustness, and efficiency, since this requires quantitative measures for each of these measures.

In addition, designing and implementing real-time systems are becoming increasingly complex because of many added features such as fancy graphical users interfaces (GUIs), visualization capabilities and the use of many sensors of different types. Therefore, many software engineering issues such as reusability and the use of COTS (Commercial Off-The Shelf) components (Profeta, 1996), reliability (Kapur *et al.*, 1996; Kim & Subbaraman, 1997; Stewart & Khosla, 1997), and embedded testing (Weller *et al.*, 1990) are now getting more attention from system developers.

In a previous paper, we proposed to use formal semantics to define performance characteristics of sensor systems (Dekhil & Henderson, 1996a). In this paper, we address these and other problems related to sensor system modeling and evaluation. We start by presenting a theoretical framework for modeling and designing sensor systems based on a formal semantics in terms of a virtual sensing machine. This framework defines an explicit tie between the specification, robustness and efficiency of the sensor system by defining several quantitative measures that characterize certain aspects of the system's behavior. Figure 1 illustrates our proposed approach which provides static analysis (e.g., time and space complexity) and dynamic handles that assist in monitoring and debugging the system.

1.1 Sensor Modeling

Each sensor type has different characteristics and functional description. Therefore it is desirable to find a general model for these different types that allows modeling sensor systems that are independent of the physical sensors used, and enables studying the performance and robustness of such systems. There have been many attempts to provide "the" general model along with its mathematical basis and description. Some of these modeling techniques concern error analysis and fault tolerance of multisensor systems (Brooks & Iyengar, 1993; Dekhil & Sobh, 1997; Iyengar & Prasad, 1994; Nadig *et al.*, 1993; Prasad *et al.*, 1991; Prasad *et al.*, 1994). Other techniques are model-based and require a priori knowledge of the scanned object and its environment (Durrant-Whyte,

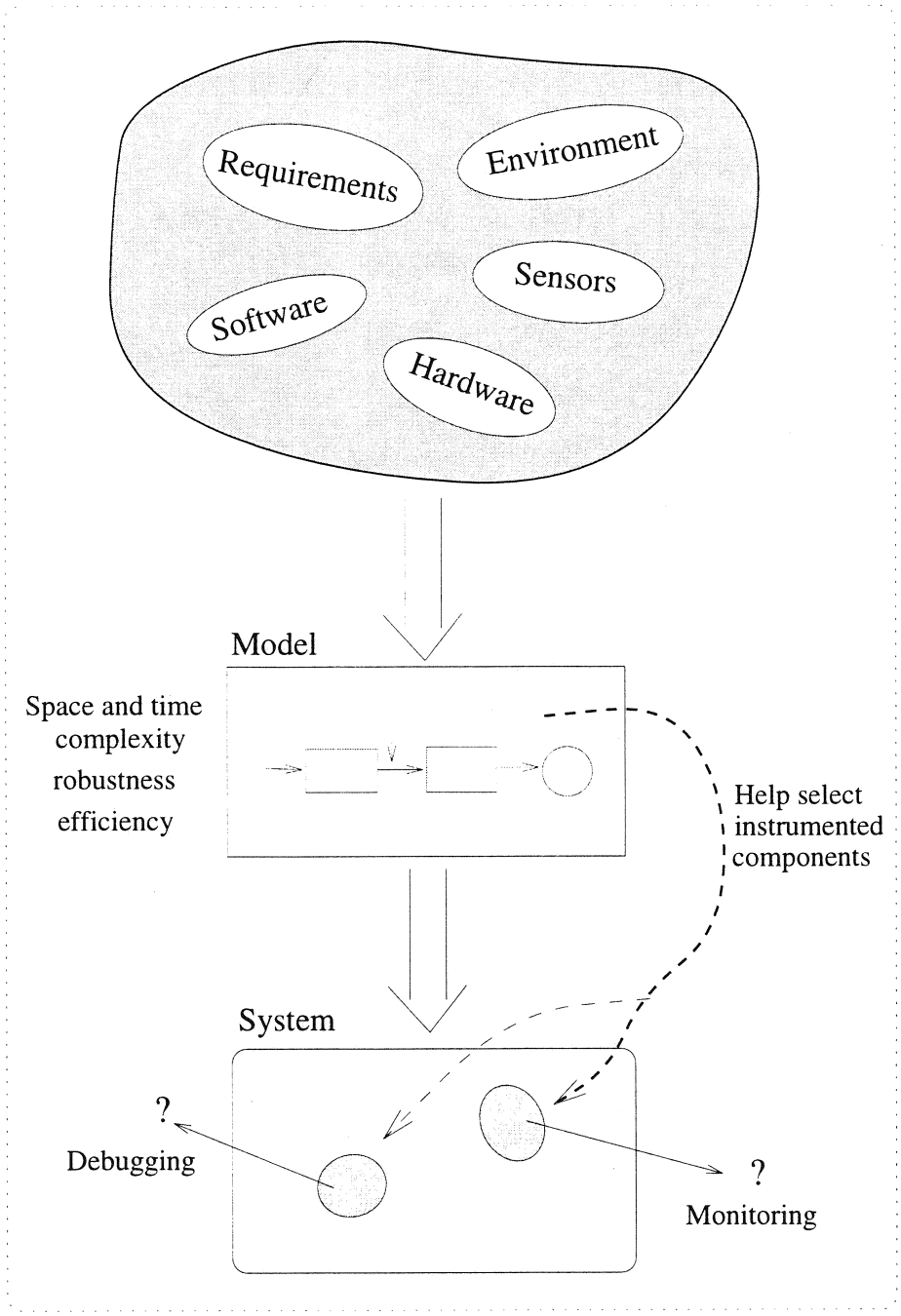


Figure 1: The proposed modeling approach.

1988; Groen *et al.*, 1993; Joshi & Sanderson, 1994). These techniques help fit data to a model, but do not provide the means to compare alternatives. Task-directed sensing is another approach to devise sensing strategies (Briggs & Donald, 1994; Hager & Mintz, 1991; Hager & Mintz, 1989), but again, it does not provide measures to evaluate the sensor system in terms of robustness and efficiency.

Another approach to modeling sensor systems is to define sensori-computational systems associated with each sensor to allow design, comparison, transformation, and reduction of any sensory system (Donald, 1995). In this approach the concept of information invariants is used to define some measure of information complexity. This approach provides a very strong computational theory which allows comparing sensor systems, reducing one sensor system to another, and measuring the information complexity required to perform a certain task. However, as stated by Donald, the measures for information complexity are fundamentally different from performance measures. Also, this approach does not permit one to judge which system is "simpler," "better," or "cheaper."

To that end, we introduce the notion of an *Instrumented Logical Sensor System* (ILSS) which represents our methodology for incorporating design tools and allows static and dynamic performance analysis, on-line monitoring, and embedded testing. Figure 2 shows the components of the ILSS. Currently, an ILSS library is under development as part of an interactive graphical programming environment called "CWave" used to design and execute real-time control systems (refer to "http://easy.cs.utah.edu/cwave/cwave.htm" for more information about the CWave project).

2 Performance Semantics of Sensor Systems

The use of sensors in safety critical applications, such as transportation and medicine, requires a high level of reliability. However, increased robustness and reliability of a multisensor system requires increased cost through redundant components and more sensor readings and computation. In contrast, increasing the efficiency of the system means less redundant components, fewer sensor readings and less computation. Performance analysis is crucial to making an informed tradeoff between design alternatives.

Performance analysis consists of a static analysis of a specification of the system and its parameters as well as a dynamic analysis of the system's run-time behavior. The static analysis can be based on some formal description of the syntax and semantics of the sensor system, while the dynamic analysis requires on-line monitoring of some quantitative measures during run-time.

Our goal is to achieve strong performance analysis and provide information which allows the user to make informed choices concerning system tradeoffs. This involves a sensor system model which permits quantitative measures of time and space complexity, error, robustness, and efficiency, and which facilitates analysis, debugging and on-line monitoring.

Formal semantics of programming languages provides techniques to describe the meaning of a

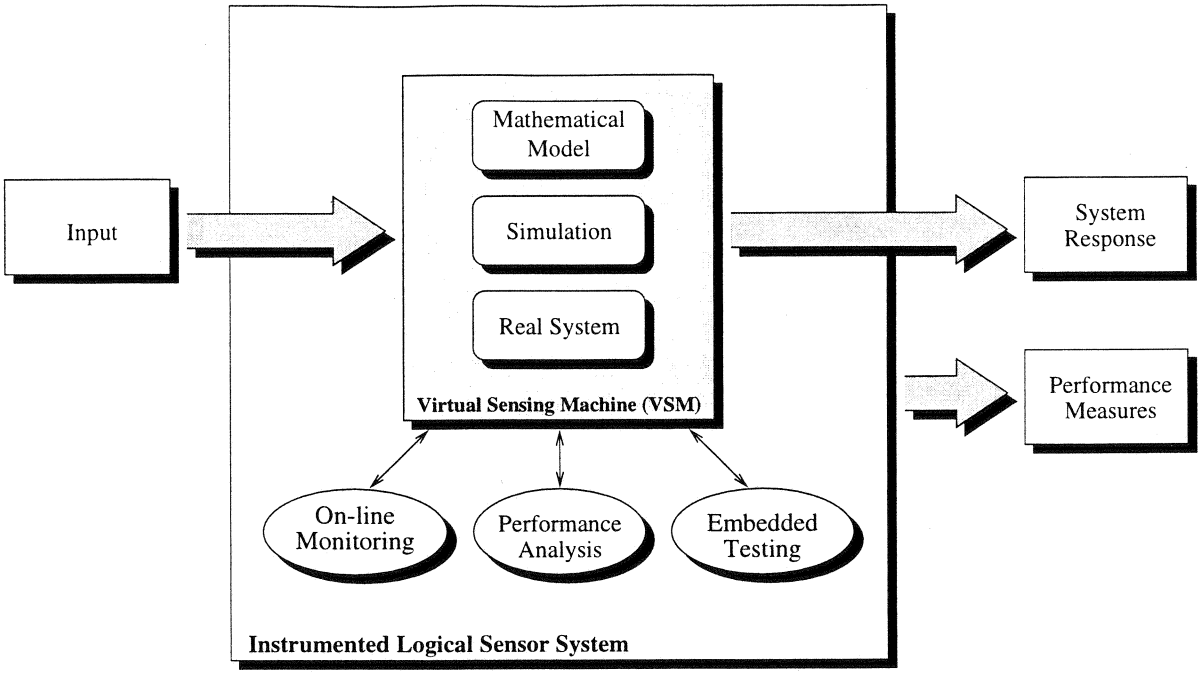


Figure 2: The Instrumented Logical Sensor System Components.

language based on precise mathematical principles. These formal techniques should provide the following: precise machine-independent concepts, unambiguous specification techniques, and a rigorous theory to support reliable reasoning (Gordon, 1979). The main types of formal semantics are: *denotational semantics* which concerns designing denotations for constructs, *operational semantics* which concerns the specification of an abstract machine together with the machine behavior when running the program, and *axiomatic semantics* which concerns axioms and rules of inference for reasoning about programs.

Our view is that performance semantics should allow us to compute measures of interest on program structures. Denotational semantics is the closest to our view since, according to (Ashcroft, 1982), to specify the semantics of a language denotationally means to specify a group of functions which assigns mathematical objects to the program and to parts of programs (modules) in such a way that the semantics of a module depends only on the semantics of the submodules. Thus, given a set of programs, $\mathcal{P}$, from a language, and an operating context, $\mathcal{C}$, the semantics is a set of functions

$$\mathcal{F} = \{f_i\}$$

where

$$f_i : \mathcal{P} \times \mathcal{C} \rightarrow \Re$$

where $\Re$ is the measurement domain.

The static semantics defines structural measures over the syntax of $p \in \mathcal{P}$. This includes standard measures such as maximum depth of the program graph, branching measures, data structure properties, storage estimates and standard computational complexity measures. Note that these can be determined without reference to $\mathcal{C}$ (i.e., $f : \mathcal{P} \rightarrow \Re$). This can be extended to include functions of the operational context $\mathcal{C}$, including sensor models, accuracy, precision, redundancy and replacement, as well as operating system effects, communication strategies and protocols, and processor properties.

The dynamic semantics include validity measures and operational characteristics. Validity measures permit the comparison of behavior models to actual run-time performance (monitors), while operational characteristics are simply measures of run-time values (taps). The values of a tap or monitor are represented as a sequence $X = (x_n : n \in \mathcal{N})$; x_n is the n^{th} value produced by the tap or monitor

$$X : \mathcal{N} \rightarrow S$$

where S is the structure produced by the tap or monitor.

2.1 Simple Example: Time Vs. Robustness Using Sonar Readings

This simple example demonstrates the motivation of the proposed approach and how it can be used to select between alternatives. In this example we have a "classical" tradeoff between speed (time to accomplish a certain task) and robustness (a combination of accuracy and repeatability). Assume that the sonar has been calibrated to eliminate any environmental effects (e.g., wall type, audio noises, etc.). The variables in this case are the accuracy of the physical sonar sensor and the number of readings taken for the same position.

Assuming the time to take one reading is t, the error standard deviation is σ, and the probability of a bad reading is Pr_b, taking one reading yields minimum time and worst accuracy. By adding a filter (e.g., averaging) and taking multiple readings, accuracy increases and time also increases. Therefore, we need quantitative measures to decide how many readings are needed to achieve the required accuracy (measured in terms of the standard deviation of the error) within a time limit.

Using the formalism presented earlier, the semantics of this problem can be defined using the set of functions $\mathcal{F} = \{time, error, repeatability\}$. In the case of using a single reading these functions can be written as:

$$time(single) = t$$

$$error(single) = \frac{\sigma}{\sqrt{(1 - Pr_b)}}$$

$$repeatability(single) = 1 - Pr_b$$

Now, if we take the average of n readings, the semantics can be written as:

$$time(average) = nt + \tau_n$$

$$error(average) = \frac{\sigma}{\sqrt{n * (1 - Pr_b)}}$$

$$repeatability(average) = 1 - Pr_b^n$$

where τ_n is the time to calculate the average of n readings, and $\tau_1 = 0$.

In this simple example we were able to get estimates of the required measures using mathematical models. However, we did not consider the changes in the environment and how it affects these measures. In this case, the set of functions $\mathcal{F}$ are mappings from the cross product of the program $\mathcal{P}$ and the operating context C to the measurement domain $\Re$, that is

$$f_i : \mathcal{P} \times \mathcal{C} \rightarrow \Re$$

To solve this problem, we either have to model the environmental effects and include it in our model, or we may need to conduct simulations if a mathematical model is not possible. Simulation is a very useful tool to approximate reality, however, in some cases even simulation is not enough to capture all the variables in the model, and real experiments with statistical analysis may be required to get more accurate results.

3 Sensor System Specification

The ILSS approach is based on *Logical Sensor Systems* (LSS) introduced by Henderson and Shilcrat (Henderson & Shilcrat, 1984). LSS is a methodology to specify any sensor in such a way that hides its physical nature. The main goal behind LSS was to develop a coherent and efficient presentation of the information provided by many sensors of different types. This representation provides a means for recovery from sensor failure and also facilitates reconfiguration of the sensor system when adding or replacing sensors (Henderson *et al.*, 1985).

We define the ILSS as an extension to the LSS and it is composed of the following components (see Figure 3):

1. *ILS Name*: uniquely identifies a module.

2. *Characteristic Output Vector (COV)*: strongly typed output structure. We have one output vector (COV_{out}) and zero or more input vectors (COV_{in}).

3. *Commands*: input commands to the module ($Commands_{in}$) and output commands to other modules ($Commands_{out}$).

4. *Select Function*: selector which detects the failure of an alternate and switches to another alternate (if possible).

5. *Alternate Subnets*: alternative ways of producing the COV_{out}. It is these implementations of one or more algorithms that carry the main functions of the module.

6. *Control Command Interpreter (CCI)*: interpreter of the commands to the module.

7. *Embedded Tests*: self testing routines which increase robustness and facilitate debugging.

8. *Monitors*: modules that check the validity of the resulting COVs.

9. *Taps*: hooks on the output lines to view different COV values.

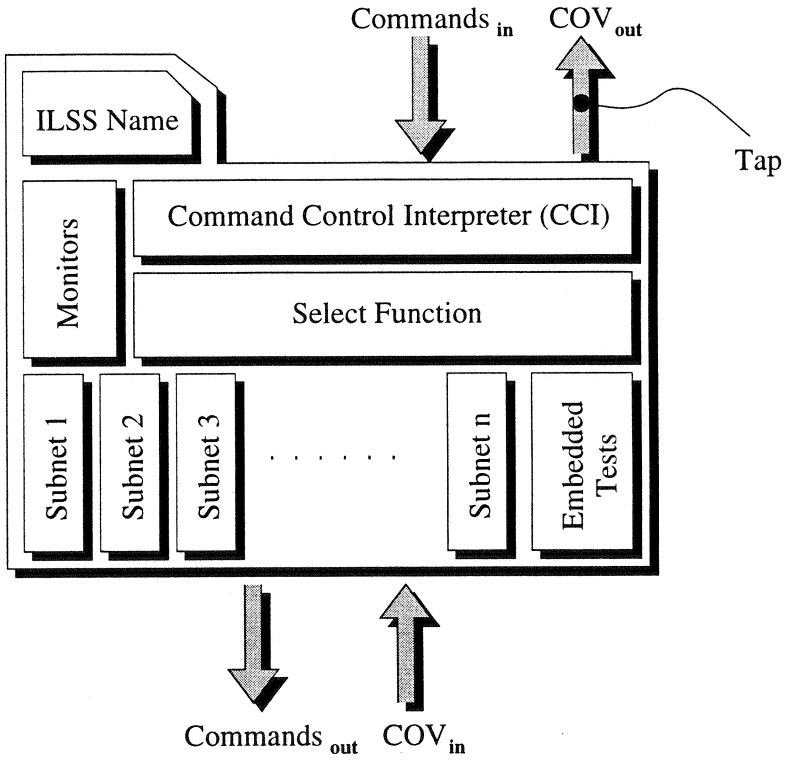


Figure 3: The extended logical sensor module.

These components identify the system behavior and provide mechanisms for on-line monitoring and debugging. In addition, they give handles for measuring the run-time performance of the system.

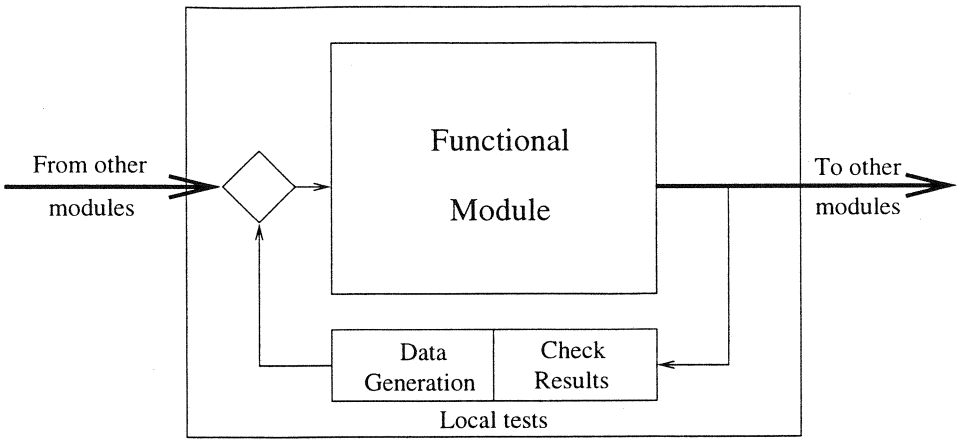


Figure 4: Local embedded testing.

Monitors are validity check stations that filter the output and alert the user to any undesired results. Each monitor is equipped with a set of rules (or constraints) that governs the behavior of the COV under different situations.

Embedded testing is used for on-line checking and debugging proposes. Weller proposed a sensor processing model with the ability to detect measurement errors and to recover from these errors (Weller *et al.*, 1990). This method is based on providing each system module with verification tests to verify certain characteristics in the measured data and to verify the internal and output data resulting from the sensor module algorithm. The recovery strategy is based on rules that are local to the different sensor modules. We use a similar approach in our framework called *local embedded testing* in which each module is equipped with a set of tests based on the semantic definition of that module. These tests generate input data to check different aspects of the module, then examine the output of the module using a set of constraints and rules defined by the semantics. Also these tests can take input data from other modules if we want to check the operation for a group of modules.

Figure 4 illustrates the idea of local embedded testing. Local embedded testing increases the robustness of the system and provides the user with possible locations to tap into when there is a problem with the system.

3.1 Construction Operators

In our proposed framework, a sensor system is composed of several ILSS modules connected together in a certain structure. We define operations for composing ILSS modules, and then define the semantics of these operations in terms of the performance parameters. Some of these operations are (see Figure 5):

- $Serial(ILSS1, ILSS2)$: two logical modules are connected in series. Here $COV3 = COV2$.
- $Select(ILSS1, ILSS2)$: $COV3$ is equal to either $COV1$ or $COV2$.
- $Combine(ILSS1, ILSS2)$: $COV3$ is the concatenation of $COV1$ and $COV2$.

For these simple constructs, the semantics is defined as a set of functions that propagate the required performance measures. Several techniques can be used for propagation. Best case analysis, worst case analysis, average, etc. Selecting among these depends on the application, hence it should be user defined. As an example, the time of the resulting logical system using worst case analysis can be calculated as follows:

- $time(Serial(ILSS1, ILSS2)) = time(ILSS1) + time(ILSS2)$
- $time(Select(ILSS1, ILSS2) = max(time(ILSS1), time(ILSS2))$
- $time(Combine(ILSS1, ILSS2) = max(time(ILSS1), time(ILSS2))$

Hence, the semantic functions of the composite system are defined in terms of the semantic functions of the subcomponents, Similarly, functions that define the propagation of other performance measures can be defined in the same way.

For error propagation, we can use a simple approach which does not require carrying a lot of information through the system. This approach is based on the uncertainty propagation described in (Faugeras, 1993; Holman & W. J. Gajda, 1978). Assume that we have a certain module with n inputs $X = (x_1, x_2, \ldots, x_n)$ and m outputs $Y = (y_1, y_2, \ldots, y_m)$ such that $Y = f(X)$, and assume that we know the error variance associated with the input vector $\Lambda_X = (\Lambda_{x_1}, \Lambda_{x_2}, \ldots, \Lambda_{x_n})$ (see Figure 6), then the error variance for the output vector can be calculated using the equation:

$$\Lambda_Y = \left(\frac{\partial Y}{\partial X}\right) \Lambda_X \left(\frac{\partial Y}{\partial X}\right)^T$$

where $\frac{\partial Y}{\partial X}$ is the partial derivative of Y with respect to X evaluated at the measured value of the input vector X. If all the elements in X are independent variables, then this equation can be written as:

$$\Lambda_{y_i} = \sum_{j=1}^{n} \left(\frac{\partial y_i}{\partial x_j}\right)^2 \Lambda_{xj} \ , i = 1, 2, \ldots, m$$

Our overall goal is to provide a tightly coupled mechanism to map high-level performance measures onto an appropriate set of monitors, tests and taps os as to provide the required information.

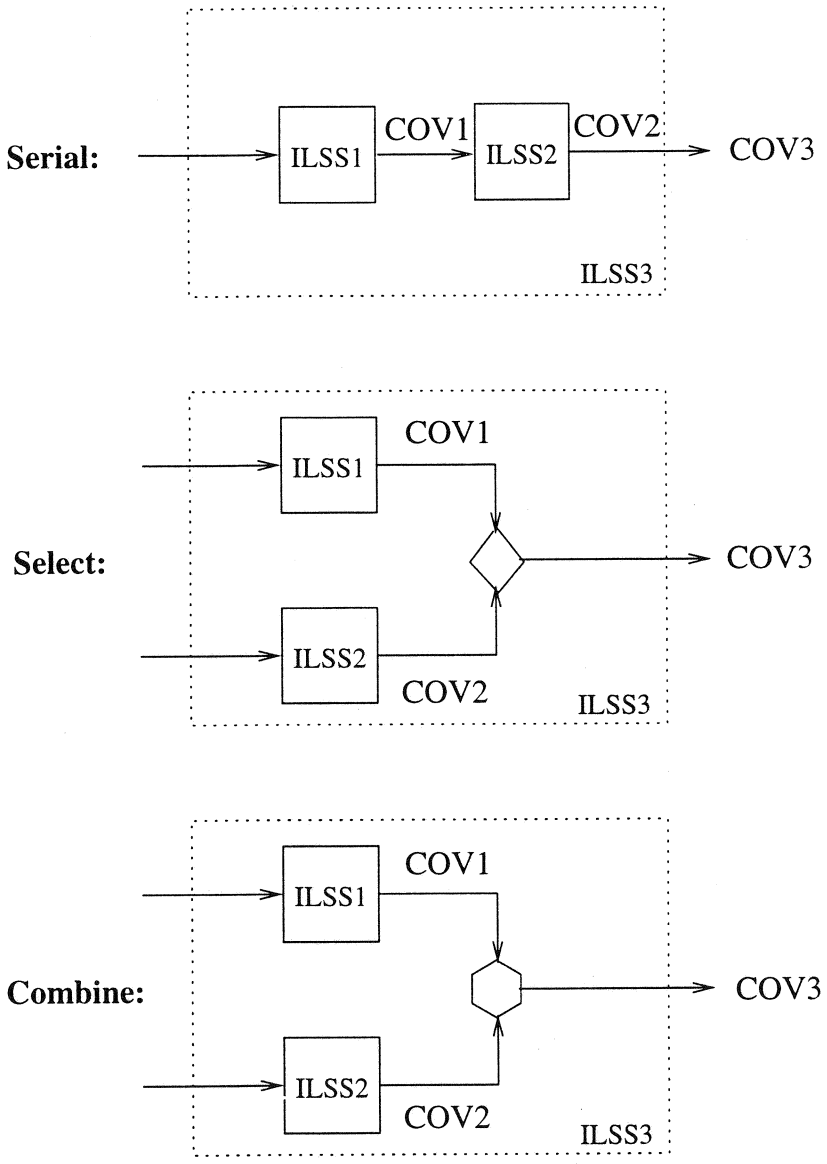


Figure 5: Some operations used for propagating the performance measures.

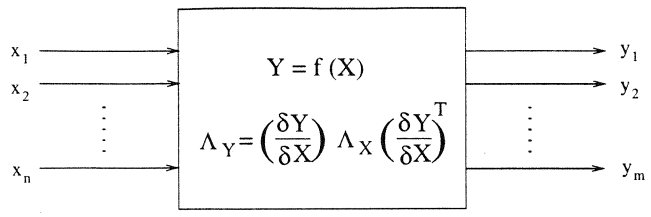


Figure 6: A simple approach for error propagation.

4 Implementation

The ultimate goal of this project is to utilize the proposed theoretical framework in a usable modeling and prototyping environment with tools for analysis, debugging, and monitoring sensor systems with emphasis on robot control applications. Thus, we are developing an ILSS library within a visual programming system called CWave targeted toward the development of control systems for measurement devices and hardware simulations. CWave is developed by the Component Software Project (CSP) research group in the Department of Computer Science at the University of Utah in cooperation with the CSP group at Hewlett Packard Research Labs in Palo Alto, California.

CWave is based on a reusable software components methodology where any system can be implemented by visually wiring together predefined and/or user created components and defining the dataflow between these components. The CWave design environment includes several important features that make it suitable to use as a framework for implementing ILSS components. Some of these features are:

- Open architecture with ease of extensibility.
- Drag-and-drop interface for selecting components.
- Several execution modes including single step, slow, and fast execution.
- On-line modification of component properties.
- The ability to add code interactively using one of several scripting languages including Visual Basic and Java Script. This is particularly useful to add monitors and/or taps on the fly.
- Parallel execution using visual threads.
- On-line context sensitive help.

Figure 7 shows the CWave design environment with some of its features.

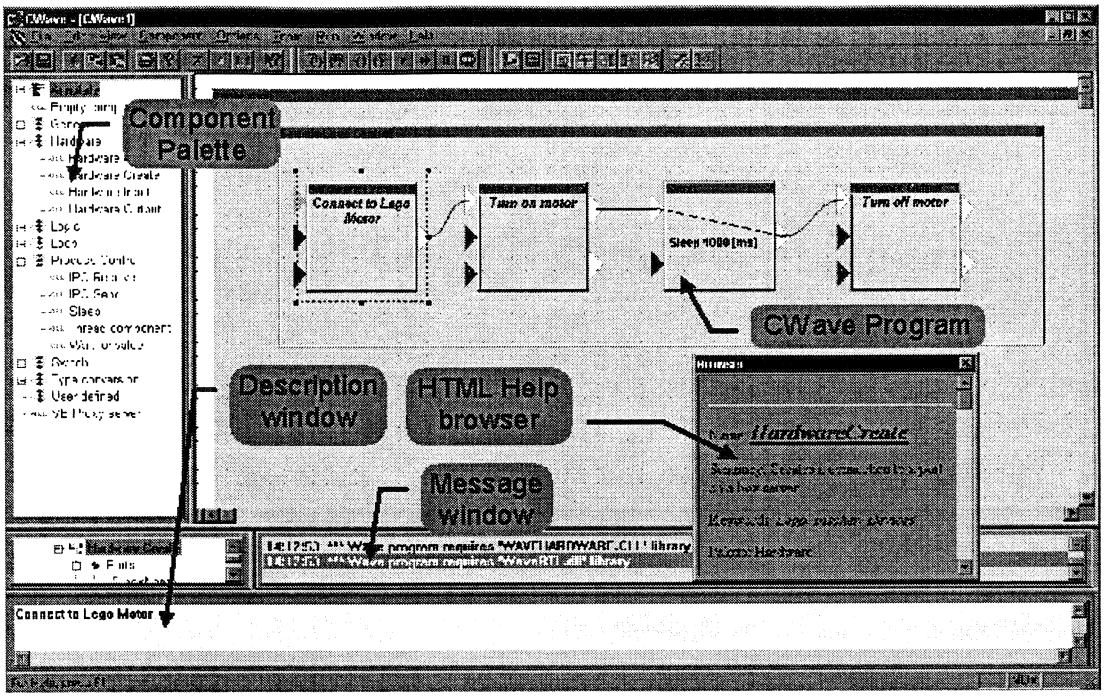


Figure 7: CWave design environment.

4.1 Implementing ILSS Components

An object-oriented approach is used to develop the ILSS components using Visual C++ for implementation. Each component is an object that possesses some basic features common to all components plus some additional features that are specific to each ILSS type. The following are some of the basic functions supported by all components:

Initialize: performs some initialization steps when the component is created.

Calibrate: starts a calibration routine.

Sense: generates the COV corresponding to the current input and the component status.

Reset: resets all the dynamic parameters of the component to their initial state.

Test: performs one or more of the component's embedded tests.

Select: selects one of the alternate subnets. This allows for dynamic reconfiguration of the system.

Monitor: observes the COV and validate its behavior against some predefined characteristic criteria.

Tap: displays the value of the required variables.

We used several design patterns in designing and implementing the components. Design patterns provide reliable and flexible object-oriented designs that can accommodate rapid modifications and extensions (Gamma *et al.*, 1995). For example, the *decorator* pattern is used to dynamically attach additional functionality to the object. This is particularly useful in our case where the user can dynamically choose the performance measures to be propagated and the values to be monitored while the system is running.

4.2 The ILSS Component Library

To hide most of the implementation details and provide higher reusability and reliability of the system, we are currently developing an ILSS library which will include some of the commonly used sensors for mobile robot applications such as cameras and sonar sensors. This library is expandable where more components can be added easily. The user can select one of several sensor types and modify its parameters and/or algorithms to fit the application. In addition, on-line monitoring and embedded testing will be defined for each component and the user can add or modify there routines to get the required information.

Components are added to the library after extensive testing and debugging to ensure reliability. The components in the library should be flexible enough to accommodate different application with varying needs. However, there is a tread-off between reusability and efficiency (see Figure 8.) Increasing the generality of a component to accommodate different applications with varying needs may require a lot of overhead which reduces the efficiency, while building a component for a specific application can be optimized to that application, but that will reduce its reusability. In conclusion, some design decisions has to be made and several factors should be considered when building new components to be added to the ILSS library.

5 Example: Wall Pose Estimation

The following example illustrates the use the proposed framework to model and analyze two alternatives for determining flat wall position and orientation using sonar sensors (Dekhil & Henderson, 1996b; Henderson *et al.*, 1996b; Henderson *et al.*, 1996a; Henderson *et al.*, 1997). The sonar sensors are mounted on a LABMATE mobile robot designed by Transitions Research Corporation. The LABMATE was used for several experiments in the Department of Computer Science at the University of Utah. It was also entered in the 1994 and 1996 AAAI Robot Competition (Schenkat

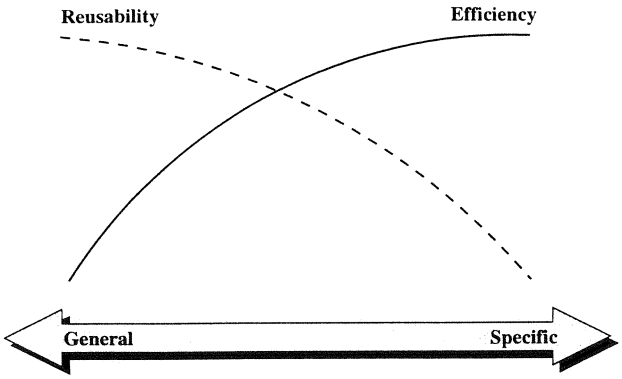


Figure 8: The tradeoffs between efficiency and usability.

et al., 1994) and it won sixth and third place , respectively. For that purpose, the LABMATE was equipped with 24 sonar sensors, eight infrared sensors, a camera and a speaker. [1] Figure 9 shows the LABMATE with its equipment.

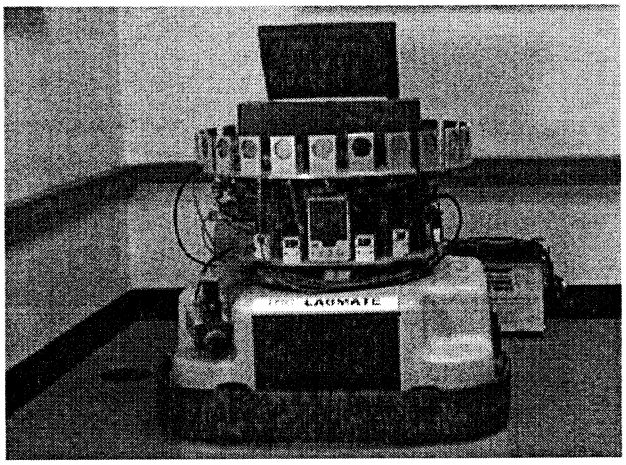


Figure 9: The LABMATE robot with its equipments.

In this example, we consider two different logical sensors to determine wall pose and find the corresponding errors and time complexity for each. The first ILSS considers the sonar sensor as a point sensor (i.e., the location of the return point is at the middle of the sonar spread wedge.) The second ILSS deals with the sonar sensor as a wedge sensor (i.e., it returns a wedge centered at the

[1] The LABMATE preparations, the sensory equipments, and the software and hardware controllers were done by L. Schenkat and L. Veigel at the Department of Computer Science, University of Utah.

sonar sensor and spread by an angle 2θ.) Both ILSS use two sonar readings to calculate the wall position. Figure 10 shows the two logical sensors.

In this figure, r_1 and r_2 are the two sonar readings generated from $Sonar1$ and $Sonar2$, respectively. $Point\_Sonar$ determines the location of the sonar reading as the mid-point of its spread wedge. $Wedge\_Sonar\_Line$ takes the two range values r_1 and r_2, and the spread angle of the sonar beam θ, and returns two 2D points on the line representing the wall.

5.1 System Modeling and Specification

As shown in Figure 10, ILSS1 is composed of four modules, two *Sonar* modules and two *Point_Sonar* modules, and a $Combine$ operator. On the other hand, LSS2 has three modules, two *Sonar* modules and a *Wedge_Sonar_Line* module preceded by a $Combine$ operator. Figure 11 shows the attributes for each of these modules.

In this figure, each ILSS is defined in terms of a set of components that characterize the module. The data and the corresponding performance measures start from the $Sonar$ module and propagate upward until they reach the COV of the main ILSS. On the other hand, the commands start from the main ILSS and propagate downward until they reach the $Sonar$ module. The COV is composed of two parts: *data* and *performance measures*. For example, COV_{out} for $Sonar1$ is

$$(\{r_1, \theta\}, \{t, \Lambda_{r1}, \Lambda_\theta\})$$

where t is the time consumed to execute the module and Λ_{r1} and Λ_θ are the error variances for r_1 and θ, respectively. In this example, each module has only one alternate subnet, therefore, the select function is trivial.

5.2 Performance Semantic Equations

Using worst case analysis, the performance semantic equations of the *time* and *error* for ILSS1 and ILSS2 can be written as:

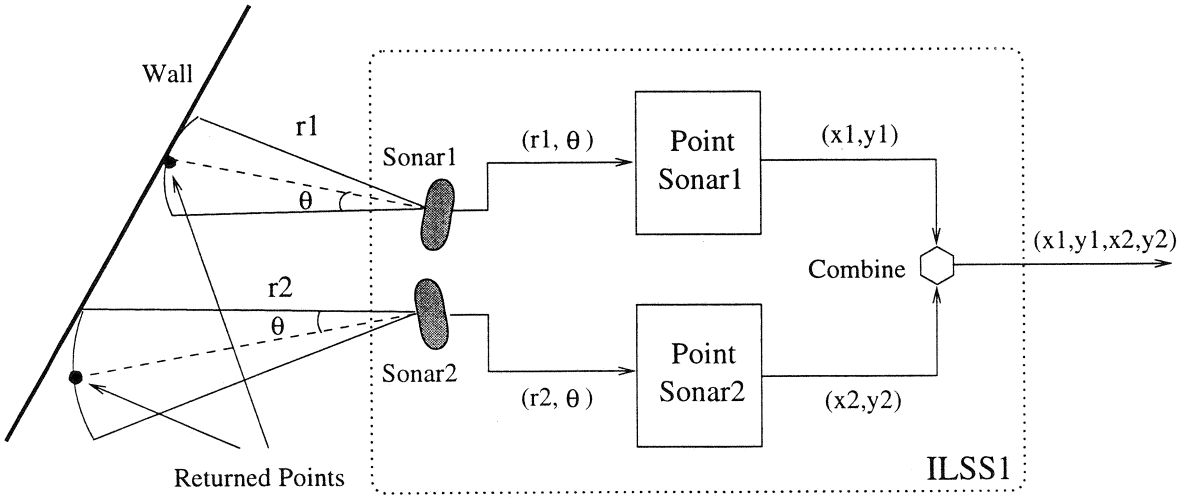
$$time(ILSS1) = time(Combine(Serial(Sonar1, Point\_sonar1), serial(Sonar2, Point\_sonar2)))$$

$$error(ILSS1) = error(Combine(Serial(Sonar1, Point\_sonar1), serial(Sonar2, Point\_sonar2)))$$

$$time(ILSS2) = time(serial(combine(Sonar1, Sonar2), Wedge\_sonar\_line))$$

$$error(ILSS2) = error(serial(combine(Sonar1, Sonar2), Wedge\_sonar\_line))$$

Now, we need to calculate the time and error for the subcomponents. Assume that t_{sonar1}, t_{sonar2}, t_{point_sonar1}, t_{point_sonar2}, and $t_{wedge\_sonar\_line}$ are the time for the subcomponents, and Λ_{r1}, Λ_{r2}, and



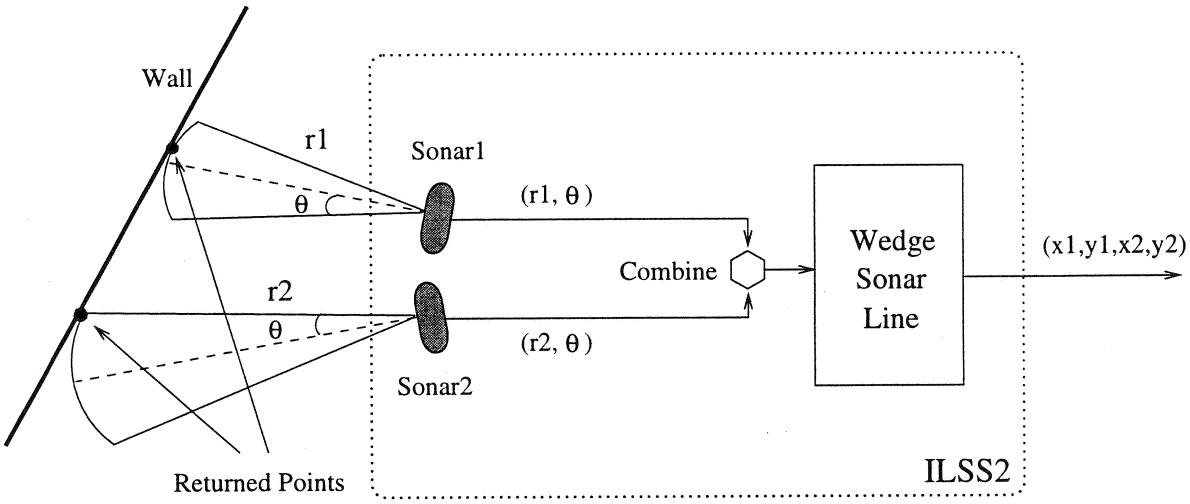


Figure 10: Two Instrumented Logical Sensors for determining wall position.

ILSS Name	Sonar	Point-Sonar	Wedge-Sonar-Line
COV in	None	$\binom{\{r,\theta\}}{\{t,\ \Lambda_r\ \Lambda_\theta\}}$	$\binom{\{r1,\theta1,r2,\theta2\}}{\{t,\Lambda_{r1},\Lambda_{\theta1},\Lambda_{r2},\Lambda_{\theta2}\}}$
COV out	$\binom{\{r,\theta\}}{\{t,\ \Lambda_r\ \Lambda_\theta\}}$	$\binom{\{x,y\}}{\{t,\ \Lambda_x\ \Lambda_y\}}$	$\binom{\{x1,y1,x2,y2\}}{\{t,\Lambda_{x1},\Lambda_{y1},\Lambda_{x2},\Lambda_{y2}\}}$
Commands in	{calib, sense}	{init, calib, sense}	{init, calib, sense, reset}
Commands out	None	{calib, sense}	{init, calib, sense}
CCI	cci_sonar()	cci_point_sonar()	cci_wedge_sonar()
Select Function	call read_sonar	call calculate_xy	call calc_line_points
Alt. Subnet	read_sonar()	calculate_xy ()	calc_line_points()
Embedded Tests	test_sonar()	test_point_sonar()	test_wedge_sonar()
Monitors	watch_sonar()	watch_point_sonar()	watch_wedge_sonar()
Taps	display_sonar()	display_point_sonar()	display_wedge_sonar()

Figure 11: The attributes of the main modules used in the example.

Λ_θ are the error measures for r_1, r_2, and θ, respectively. The time for LSS1 and LSS2 can be easily calculated using the propagation operations discussed earlier as follows:

$$time(ILSS1) = max(t_{sonar1} + t_{point\_sonar1}, t_{sonar2} + t_{point\_sonar2})$$

$$time(ILSS2) = max(t_{sonar1}, t_{sonar2}) + t_{wedge\_sonar\_line}$$

where P_{ILSS1} and P_{ILSS2} represent the current program steps executed on ILSS1 and ILSS2, respectively.

Propagating the error requires more elaborate analysis for each component. At the first level, we have the physical sonar sensor where the error can be determined either from the manufacturer specs, or from experimental data. In this example we will use the error analysis done by Schenkat and Veigel (Schenkat *et al.*, 1994) in which there is a Gaussian error with mean μ and variance σ^2. From this analysis, the variance is a function of the returned distance r. To simplify the problem let's assume that the variance in both sensors is equal to $4.0mm^2$. Also, in this logical sensor there is an algorithmic error due to the fact that the actual point can be anywhere on the sonar wedge, and not just at the center as is returned. This error can be calculated by assuming that θ is a random variable uniformly distributed over the range $(-11^o, 11^o)$ given that the beam spread of the sonar is 22^o (see Figure 12). The variance of the angle θ is therefore equal to:

$$\Lambda_\theta = \frac{(22\pi/180)^2}{12} = 0.0123rad^2$$

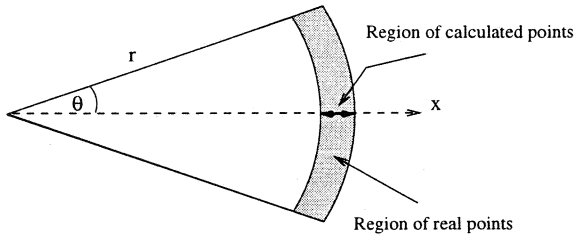


Figure 12: The error in the point-sonar module.

In this case we have:

$$error(Sonar1) = \{4.0mm^2, 0.0123rad^2\}$$

$$error(Sonar2) = \{4.0mm^2, 0.0123rad^2\}$$

Next, for the $Point\_Sonar$, there are two sources of errors; error due to the uncertainty in r, and the algorithmic error represented by the variance in θ. The output of this module is the estimated (x, y) location of the point where the sonar beam hits the wall. This can be written as:

$$x_1 = r_1 \cos\theta, \; y_1 = r_1 \sin\theta$$

$$x_2 = r_2 \cos\theta,\ y_2 = r_2 \sin\theta$$

Since r_i and θ are independent, then the error associated with x_i and y_i can be calculated as follows:

$$\Lambda_{x_i} = \left(\frac{\partial x_i}{\partial r}\right)^2 \Lambda_r + \left(\frac{\partial x_i}{\partial \theta}\right)^2 \Lambda_\theta$$

$$\Lambda_{y_i} = \left(\frac{\partial y_i}{\partial r}\right)^2 \Lambda_r + \left(\frac{\partial y_i}{\partial \theta}\right)^2 \Lambda_\theta$$

To calculate the worst case error, we evaluate these expressions at $\theta = 11^o$, assuming that $r_1 = 1000mm$ and $r_2 = 1100mm$, we get:

$$\Lambda_{x_1} = \cos^2\theta * 4.0 + (-r_1 \sin\theta)^2 * 0.0123 = 451.67mm^2$$

$$\Lambda_{y_1} = \sin^2\theta * 4.0 + (r_1 \cos\theta)^2 * 0.0123 = 11852.18mm^2$$

$$\Lambda_{x_2} = \cos^2\theta * 4.0 + (-r_2 \sin\theta)^2 * 0.0123 = 545.72mm^2$$

$$\Lambda_{y_2} = \sin^2\theta * 4.0 + (r_2 \cos\theta)^2 * 0.0123 = 14341.28mm^2$$

Now, we can write the error functions for Point_Sonar modules as follows:

$$error(Point\_Sonar1) = \{451.67mm^2, 11852.18mm^2\}$$

$$error(Point\_Sonar2) = \{545.72mm^2, 14341.28mm^2\}$$

and for ILSS1 we have:

$$error(ILSS1) = \{451.67mm^2, 11852.18mm^2, 545.72mm^2, 14341.28mm^2\}$$

In the $Wedge\_Sonar\_Line$ module, there are five possible cases for that line depending on the values of r_1 and r_2 (Henderson *et al.*, 1996a). In any case, the two points laying on the line can be written as:

$$x_1 = r_1 \cos\alpha_1,\ y_1 = r_1 \sin\alpha_1$$

$$x_2 = r_2 \cos\alpha_2,\ y_2 = r_2 \sin\alpha_2$$

where the values of $\alpha 1$ and α_2 are between $-\theta$ to θ (see Figure 13).

Considering the worst case error, we can set $\alpha_1 = \alpha_2 = \theta$. Assuming that the error in θ is zero, then the error in the calculated points is:

$$\Lambda_{x_i} = \left(\frac{\partial x_i}{\partial r}\right)^2 \Lambda_r$$

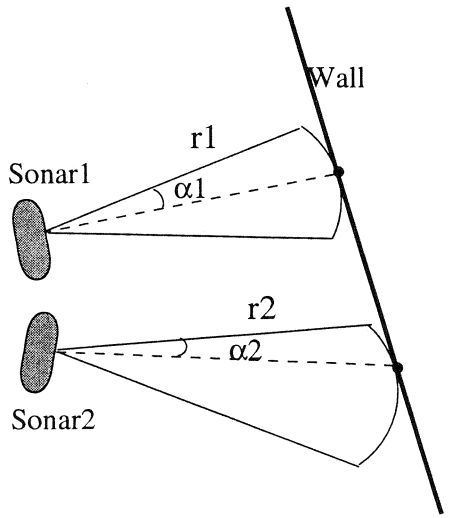


Figure 13: The general case for the points returned by the wedge_sonar_line.

$$\Lambda_{y_i} = \left(\frac{\partial y_i}{\partial r}\right)^2 \Lambda_r$$

Evaluating these expressions using the same values we used for $LSS1$ we get:

$$\Lambda_{x_1} = \cos^2\theta * 4.0 = 3.85mm^2$$

$$\Lambda_{y_1} = \sin^2\theta * 4.0 = 0.15mm^2$$

$$\Lambda_{x_2} = \cos^2\theta * 4.0 = 3.85mm^2$$

$$\Lambda_{y_2} = \sin^2\theta * 4.0 = 0.15mm^2$$

Finally, the error function for $ILSS2$ is:

$$error(ILSS2) = \{3.85mm^2, 0.15mm^2, 3.85mm^2, 0.15mm^2\}$$

Comparing the error and time measures for the ILSS1 and ILSS2 we can select between these two alternatives based on the application requirements.

This example illustrates the importance and usefulness of the ILSS library since all these analysis can be performed once and put in the library for reuse and the user does not have to go through these details again. For example, if different sonar sensor is used, then the same error analysis can be used by supplying the sensor's error variance. In addition, given that the error range has been determined, redundancy can be added using different sensor pairs to sense the same wall and a monitor can be added to detect error discrepancies.

5.3 Simulation Results

These two instrumented logical sensors were used with the LABMATE to find the location of walls using synthetic and real data. Figure 14 shows the error in both cases using synthetic data representing 300 lines at different angles from the sonars, the results show that the error using $ILSS2$ was much smaller than the error using $ILSS1$ which supports the analysis conducted using our proposed approach. In this figure, the error in ρ and θ varies based on the region to which the line belongs.[2] For example, we can see transition at line 100 which corresponds to moving from region 1 to region 2. for the results of applying these two logical sensors to run the LABMATE with more detailed comparisons see (Henderson *et al.*, 1997).

Note that, to keep things simple, we did not consider the error in the sonar location and orientation. However, these errors can be incorporated in the model in the same manner.

6 Conclusions

In this paper we presented a theoretical framework for sensor modeling and design based on defining the performance semantics of the system. We introduced the notion of *instrumented sensor systems*, which is a modeling and design methodology that facilitates interactive, on-line monitoring for different components of the sensor system. It also provides debugging tools and analysis measures for the sensor system. The instrumented sensor approach can be viewed as an abstract sensing machine which defines the semantics of sensor systems. This provides a strong computational and operational engine that can be used to define and propagate several quantitative measures to evaluate and compare design alternatives. The implementation of this framework within the CWave system was described and examples of using this framework were presented.

Currently, we are working on building an ILSS library with several design tools which will assist in rapid prototyping of sensor system and will provide an invaluable design tools for monitoring, analyzing and debugging sensor systems.

Acknowledgment

We would like to thank Professor Robert Kessler and Christian Mueller for providing the CWave program that we used to implement the instrumented sensor library. We also would like to thank professor Gary Lindstrom for his helpful input.

[2] There are five different regions that a line representing the wall can fall in. See (Henderson *et al.*, 1996a) for more details.

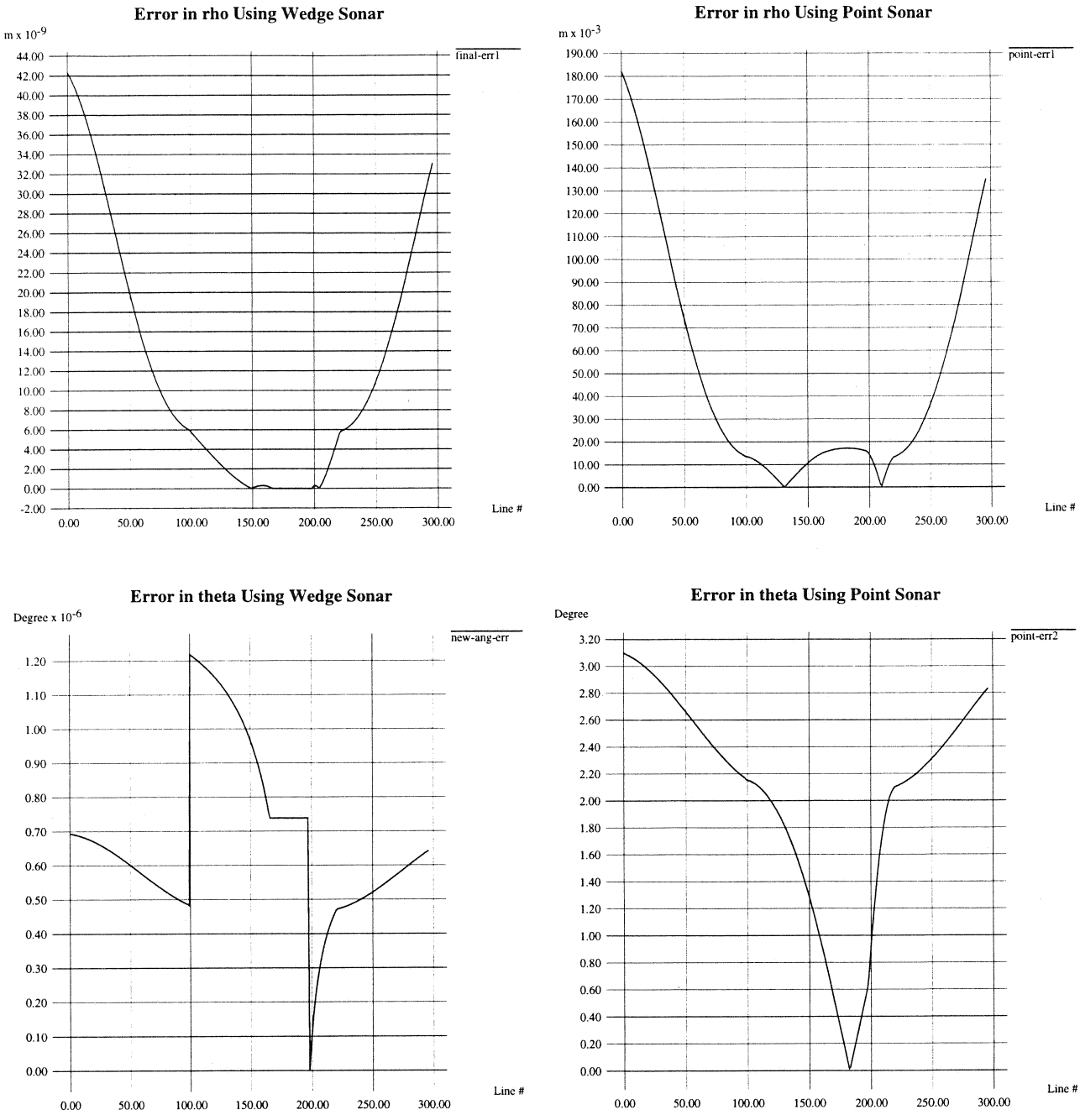


Figure 14: The error in both cases using synthetic data.

References

Ashcroft, E. A. 1982. R for semantics. *ACM Transactions on Programming Languages and Systems*, **4**(2), pp. 283–295.

Briggs, A.J., & Donald, B.R. 1994 (May). Automatic sensor configuration for task-directed planning. *Pages 1345–1350 of: IEEE Int. Conf. Robotics and Automation.*

Brooks, R. R., & Iyengar, S. 1993. *Averaging algorithm for multi-dimensional redundant sensor arrays: resolving sensor inconsistencies.* Tech. rept. Louisiana State University.

Dekhil, M., & Henderson, T. C. 1996a (December). Instrumented Sensor Systems. *Pages 193–200 of: IEEE International Conference on Multisensor Fusion and Integration (MFI 96), Washington D.C.*

Dekhil, M., & Henderson, T. C. 1996b (December). Optimal Wall Pose Determination in a Shared-Memory Multi-Tasking Control Architecture. *Pages 736–741 of: IEEE International Conference on Multisensor Fusion and Integration (MFI 96), Washington D.C.*

Dekhil, M., & Sobh, T. M. 1997 (January). Embedded tolerance analysis for sonar sensors. *In: Invited paper to the special session of the 1997 Measurement Science Conference* Measuring Sensed Data for Robotics and Automation *Pasadena, California.*

Donald, B. R. 1995. On information invariants in robotics. *Artificial Intelligence*, pp. 217–304.

Durrant-Whyte, H. F. 1988. *Integration, coordination and control of multisensor robot systems.* Kluwer Academic Publishers.

Faugeras, O. 1993. *Three-dimensional computer vision - a geometric viewpoint.* The MIT Press.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. 1995. *Design patterns: elements of reusable object-oriented software.* Addison Wesley.

Gordon, M. J. C. 1979. *Denotational description of programming languages.* Springer-Verlag.

Groen, F. C. A., Antonissen, P. P. J., & Weller, G. A. 1993. Model based robot vision. *Pages 584–588 of: IEEE Instrumentation and Measurment Technology Conference.*

Hager, G., & Mintz, M. 1991. Computational Methods for Task-directed Sensor Data Fusion and Sensor Planning. *Int. J. Robotics Research*, **10**(4), pp. 285–313.

Hager, G. D., & Mintz, M. 1989. Task-directed multisensor fusion. *In: IEEE Int. Conf. Robotics and Automation.*

Henderson, T. C., & Shilcrat, E. 1984. Logical Sensor Systems. *Journal of Robotic Systems*, Mar., pp. 169–193.

Henderson, T. C., Hansen, C., & Bhanu, B. 1985. The Specification of distributed Sensing and Control. *Journal of Robotic Systems*, Mar., pp. 387–396.

Henderson, T. C., Dekhil, M., Bruderlin, B., Schenkat, L., & Veigel, L. 1996a (February). Flat surface recovery from sonar data. *Pages 995–1000 of: DARPA Image Understanding Workshop.*

Henderson, T. C., Bruderlin, B., Dekhil, M., Schenkat, L., & Veigel, L. 1996b (April). Sonar sensing strategies. *Pages 341–346 of: IEEE Int. Conf. Robotics and Automation.*

Henderson, T. C., Dekhil, M., Bruderlin, B., Schenkat, L., & Veigel, L. 1997. Wall reconstruction using sonar sensors. *To appear in the IEEE International Journal of Robotics Research.*

Holman, J. P., & W. J. Gajda, Jr. 1978. *Experimental methods for engineers.* McGraw-Hill.

Iyengar, S. S., & Prasad, L. 1994. A general computational framework for distributed sensing and falt-tolerant sensor integration. *IEEE Trans. Systems Man and Cybernetics*, May.

Joshi, R., & Sanderson, A. C. 1994 (May). Model-based multisensor data fusion: a minimal representation approach. *In: IEEE Int. Conf. Robotics and Automation.*

Kapur, R., Williams, T. W., & Miller, E. F. 1996. System testing and reliability techniques for avoiding failure. *IEEE Computer*, November, pp.28–30.

Kim, K. H., & Subbaraman, C. 1997. Fault-tolerant real-time objects. *Communications of the ACM*, **40**(1), pp.75–82.

Nadig, D., Iyengar, S. S., & Jayasimha, D. N. 1993. New architecture for distributed sensor integration. *In: IEEE SOUTHEASTCON Proceedings.*

Prasad, L., Iyengar, S. S., Kashyap, R. L., & Madan, R. N. 1991. Functional characterization of fault tolerant integration in distributed sensor networks. *IEEE Trans. Systems Man and Cybernetics*, September, pp. 1082–1087.

Prasad, L., Iyengar, S. S., Rao, R. L., & Kashyap, R. L. 1994. Fault-tolerence sensor integration using multiresolution decomposition. *The American Physical Society*, April, pp. 3452–3461.

Profeta, J. A. 1996. Safety-critical systems built with COTS. *IEEE Computer*, November, pp.54–60.

Schenkat, L., Veigel, L., & Henderson, T. C. 1994 (Dec.). *EGOR: Design, Development, Implementation – An Entry in the 1994 AAAI Robot Competition*. Tech. rept. UUCS-94-034. University of Utah.

Stewart, D. B., & Khosla, P. K. 1997. Mechanisms for detecting and handling timing errors. *Communications of the ACM*, **40**(1), pp.87–93.

Weller, G. A., Groen, F. C. A., & Hertzberger, L. O. 1990. A sensor processing model incorporating error detection and recoverry. *Pages 351–363 of: Traditional and non-traditional robotic sensors.* Edited by T. C. Henderson. Springer-Verlag.