Sensor Explication: Knowledge-Based Robotic Plan Execution through Logical Objects

John Budenske and Maria Gini

Abstract— **Complex robot tasks are usually described as high-level goals, with no details on how to achieve them. However, details must be provided to generate primitive commands to control a real robot. A sensor explication concept, that makes details explicit from general commands, is presented.**

We show how the transformation from high-level goals to primitive commands can be performed at execution time and we propose an architecture based on reconfigurable objects that contain domain knowledge and knowledge about the sensors and actuators available. Our approach is based on two premises: (1) plan execution is an information gathering process where determining what information is relevant is a great part of the process; and (2) plan execution requires that many details are made explicit.

We show how our approach is used in solving the task of moving a robot to and through an unknown, and possibly narrow, doorway where sonic range data is used to find the doorway, walls, and obstacles. We illustrate the difficulty of such a task using data from a large number of experiments we conducted with a real mobile robot. The laboratory results illustrate how the proper application of knowledge in the integration and utilization of sensors and actuators increases the robustness of plan execution.

Keywords— **plan execution for robots, knowledge based sensing, object-oriented design.**

I. Introduction

Today's system designers are addressing more and more complex problems to solve applications such as robotics [1], [2], [3], adaptive intelligent systems [4], [5], [6], and manufacturing control systems [7].

These are challenging domains because the environment is dynamic (objects can move in unpredictable ways, and there are multiple independent entities), sensor data are noisy, there are multiple and sometimes conflicting goals, and decisions must be made in real-time. This requires a tight but flexible coupling of perception to action and the ability to take advantage of opportunities that appear in the environment.

The real-time requirement and the unpredictability of the environment make it impossible for a system to obtain complete knowledge of the environment such as to enable globally optimal decisions. So, actions must be selected using only limited, mostly local information. The major difficulty is in controlling and coordinating all the components of the system (sensors, sensor processing units, decision units, actuators, and actuator drivers) so that the global goals are achieved using the resources available.

Detecting achievement of goals is difficult. This is further hampered when goals already achieved become unachieved because of other actions (a stack of blocks might fall down because of a movement of a robot) or because of uncontrollable changes in the environment (a door was unlocked but someone locks it while the robot is inspecting another part of the building so that the robot cannot open it).

In this paper we describe a coherent approach to the execution of plans which allows specification, coordination, and control of a variety of sensors and actuators in a flexible architecture that we call a *Logical Sensor/Actuator (LSA)*. The implementation of our solution resulted in an object oriented system called the *Logical Sensor/Actuator Testbed*. The testbed contains a library of reusable and reconfigurable sensor and actuator processing entities. The design is easy to extend to other sensors or robots and to adapt to simulation systems.

We have focused our investigation on a particular domain where the robot is given a high-level detail-less plan, such as "move through a doorway", that has to be executed in a changing and only partially known environment. Many characteristics in the environment (unknown information, noisy data, etc.) provide variability. Variability not only provides difficulties that must be overcome, but also opportunities to apply knowledge and develop strategies. The task requires precision beyond that of any given single sensor, and requires the combination and interaction of multiple sensors, so providing a rich domain of interaction, abstraction, and application of knowledge.

II. The Application: Robot, Sensors, Actuators, and Knowledge

A large part of our civilized world is unstructured and dynamic. This fact has been one of the leading obstacles to mass-quantity introduction of robots into everyday life. Though there are factories and buildings in which highly structured environments have been introduced for easy integration of robots, the ratio of unstructured to structured environments will always be high. Thus, society's utilization of robotics will depend upon the ability of robots to move safely in these unstructured environments.

The variability of the environment makes it impractical to develop very detailed plans of actions before execution. The environment might change before execution begins and

This work was funded in part by NSF under grant NSF/CDA-9022509, and by the AT&T Foundation.

John Budenske is with Architecture Technology Corporation. The research described in this paper was performed while he was with the Department of Computer Science, University of Minnesota.

Maria Gini is with the Department of Computer Science, University of Minnesota, Minneapolis, MN 55455. E-mail: gini@cs.umn.edu

thus invalidate the plan. Attempting to determine, prior to execution, everything needed to achieve a goal across the domain of goals, environment states, and sensor availability would be an endless task. It makes more sense to develop, before execution, a high-level plan of what generally should be accomplished, and then generate the necessary explicit details during execution.

Thus, we are faced with a problem: how can a robot accomplish its goals and still remain reactive to its environment? Many approaches have been proposed to make robots reactive [8], [9] and to combine planning with reactivity [1], [10], [11].

Our research has focused on determining what knowledge about sensors, actuators, and processes is necessary for their effective utilization and how to use this knowledge to dynamically reconfigure the flow of data from sensors into actuators. This process of monitoring sensors, collecting data, and using the data to further guide the robot through execution must, at the same time, remain reactive to the external world and also achieve high-level goals. We call this process *sensor explication*[1] which we define as *generating from high-level commands explicit and detailed commands for the sensor and actuators.*

Instead of "directed control from a prespecified plan" the process could be described as "directive reaction to environmental changes during pursuit of a high-level goal". The central research question is determining how to transform a given "goal" into an explicit representation of "what is necessary to achieve the goal".

Suppose a mobile robot is given the task of finding a doorway and passing through it. The task can be decomposed into the sub-tasks of finding the doorway, tracking it during approach, and then passing through it [12].

Suppose the sensors available to the robot are ultrasonic sensors, placed all around the robot to give it a 360° coverage of the surrounding environment. Finding the doorway is more than just looking for a "sonic hole" signifying no object reflections in the sensor stream. The pattern of a short distance, long distance, short distance (or variation thereof) can be attributed to many natural indoor structures as well as to the general occurrence of noise. The real problem comes when the robot finds something that looks like a doorway. How can the robot verify that it is indeed a doorway? To illustrate this point, let us first play a game of "Find the Doorway" (please don't look at the even numbered figures yet!).

Figures 1, 3, 5 show the result of collecting a snapshot of ultra-sonic readings from a ring of 24 sensors. The data are collected as an array of 24 readings (x, y, θ, *distance*), where x and y are the position of the center of the sensor ring on the robot, θ is the angle from the front of the robot to the direction of the sensor, and *distance* is the returned range calculated from the time-of-flight to the object off which the sonar beam is bouncing.

Sonar data are illustrated by drawing the robot (the black square, where the notch indicates the robot's front), and plotting for each sensor an arc at the corresponding angle and range at which reflections appear to bounce off objects. An arc is used since any object in the sensor's cone (30°) can return the sound signal[2]. To further illustrate the data, a line is drawn from the midpoint of each arc to the next. The resulting outline is a visualization of the "open" space around the robot.

Now, examine this visualization and "guess" the location of the doorway. Once you have decided, look at Figures 2, 4, 6, which have a layout of the room superimposed on the original visualization. The game for Figure 1 is easy, but Figures 3 and 5 show how the doorway can be

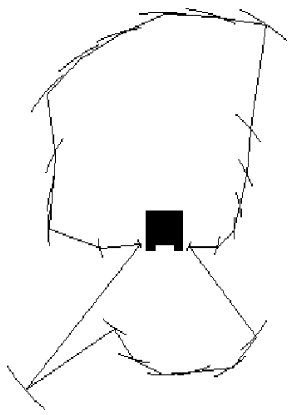

Fig. 1. Sonar data for Case 1. The robot is in a doorway with objects at various distances. The sonar returns are shown as arcs connected by lines through their centers.

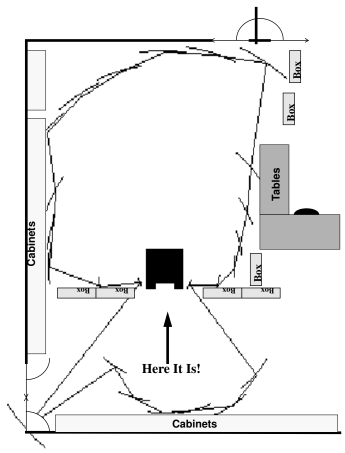

Fig. 2. Sonar data for Case 1 superimposed on a map of the room. The data match closely the walls and furniture in the room and the doorway is easy to find.

[1]from the Webster's dictionary – noun: (origin 1531) 1: to give a detailed explanation of 2: to develop the implications of: analyze logically

[2]Actually, the sensor beam's footprint is far more complex than a cone, but for this illustration, the simplicity of a cone allows for better understanding.

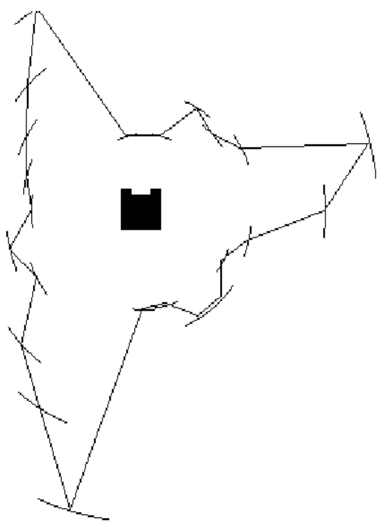

Fig. 3. Sonar data for Case 2. The robot is facing a wall with a doorway slightly to the right of the robot. The doorway is barely visible in the sonar data.

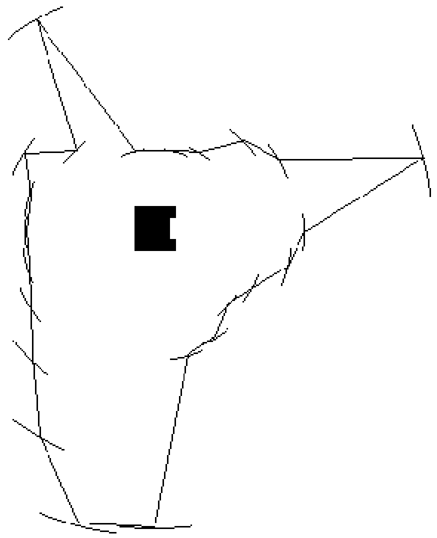

Fig. 5. Sonar data for Case 3. The robot is facing an opening.

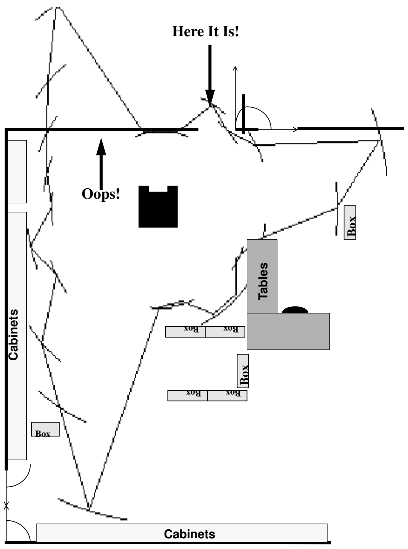

Fig. 4. Sonar data for Case 2 superimposed on a map of the room. The doorway is barely visible in the sonar data. A ghost door, caused by reflections, appears to the left front of the robot. The openings to the right and in the back of the robot could be mistaken for doorways.

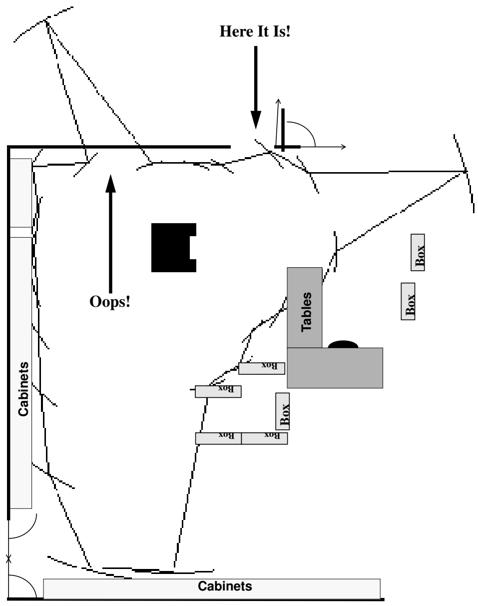

Fig. 6. Sonar data for Case 3 superimposed on a map of the room. There is an opening in front of the robot. A ghost door, caused by reflections, appears to the left rear of the robot. The doorway on the left side of the robot is practically invisible. The openings in front and to the right of the robot could be mistaken for doorways.

missed. Unfortunately, the sensed data are not always as clean as in Figure 1. The last set of figures shows the effect of physical interaction between the sensor beam and surrounding structures. This "noise" (that we call a *ghost door*) was consistently present in our experiments until the robot moved, changing its physical relationship with the environment. Of course moving the robot is not always desirable, since it increases its positional error and might interfere with other parts of the task.

Detecting the doorway is just the beginning. When tracking the doorway, it is easy to lose it and find it again later. If this occurs, is the robot sensing a ghost door or a real one? Even if the robot could sense exactly where the doorway is, moving to and through it can cause other errors. Suppose the bumpers make contact with something. Is this conclusive evidence that there is no doorway? Or does it only mean that there is something between the robot and the doorway? What does it mean to have the bumper contact something when the robot should be passing through the doorway? Is the robot misaligned to the opening? Is it farther through the opening than previously thought? Is the doorway too small for passage? Is the door in the way? Or is there an object obstructing the pathway?

One possible solution would be to use more accurate sensors and more precise dead-reckoning. However, no matter how accurate the measurements are, there will always be situations in which they are not sufficiently accurate, and increasing accuracy in general comes at additional cost that might not be justified by the application.

Overall, there are a number of things that could happen or go wrong as the robot searches for, tracks, and maneuvers through the doorway. Each situation calls for a different response, and the real problem is not so much what the response strategy should be, but what the situation is. Often extensive interaction with the environment is necessary just to collect information for assessing the situation.

Expecting the execution to always go right is futile. Errors will occur, and unexpected events will get in the way of success. Gat [13] uses the concept of "cognizant failure", or failures which can be monitored for, detected, and thus recovered from. Monitoring for errors is actually monitoring for deviations from the normal expected processing. Thus, design for cognizant failure must lead to design for "anticipatory deviation". Not only must you realize that something failed, but the control must anticipate the deviation from the norm and be ready to resolve it. Knowledge on what and when to monitor, and on how to resolve discrepancies is key to our approach.

Our experiences have led us to the conclusion that as the robot tackles each task and subtask, the situations encountered cannot simply be addressed by a single algorithm. Multiple algorithms, multiple sensing, and multiple strategies appear to be the course of action.

III. Conceptual Explanation: Sensor Explication

To be executed, a task must explicitly specify how to utilize the sensors and actuators. For the same goal, different environment situations may require different sensors, different programming and control of the sensors, and different strategies. Because this transformation process depends on the current situation it must occur at execution time. Essentially, Sensor Explication requires:

1. Identifying the information needed for execution. Sensor Explication must deal with the question of "what information is relevant and thus is needed in order to execute this goal?". The first step in our approach is to use domain knowledge to abstract from the given goal its information needs. To put it another way, the goal "achieve X" is replaced by explicit knowledge on "how to achieve X" and "how to interact with the sensors and actuators which provide the information needed to achieve X". This is what we call *Relevant-Information-Need.* This requires:
   (a) determining the information necessary to accomplish the goal. This depends on the current state, the availability of sensors and actuators, and the goal informational needs;
   (b) decomposing the goal into sub-goals. Sensor Explication must capture the relationship between goals, current situations, and information needs. The choice of how to decompose a goal into sub-goals is contingent on the current situation and often requires additional information which might have to be collected before the decomposition can be completed.
2. Finding and controlling sources of information. Sensor Explication must deal with the question of "what details of using sensors, actuators, and processes are necessary to execute this goal?" In our approach the knowledge on "how to achieve X" is coupled with knowledge about sensors, actuators, and processes, and knowledge about what parameters will satisfy the information need This is what we call *Utilization-Detail.* This is accomplished by:
   (a) selecting the source of information and collecting it. Executing primitive commands on sensors or actuators allows the system to collect the relevant information;
   (b) detecting and resolving conflicts between sub-goals. Some minor conflicts (such as contention for the same resource) can be resolved locally with simple scheduling, others require more complicated reasoning and even changing the way a sub-goal is achieved.
   (c) activating a feedback control loop to control sensors and actuators. A changing environment requires the system to be continuously reactive. This forces the system to constantly collect information and integrate it back into the execution process and continuously reselect appropriate Utilization-Detail values.
   (d) monitoring to detect goal accomplishment and error conditions. Information relevant to determining goal accomplishment and error conditions is determined first. Then that information is monitored for the occurrence of the anticipated situations.

We have utilized the concept of Logical Sensors. A Logical Sensor [14] contains both a declarative description of the sensor and procedures to control it. Logical Sensors are abstracted views of sensors and sensor processing, much like how logical I/O is used to insulate the user from the differences of I/O devices and operating systems. The specifics of the implementation can change without affecting the symbolic-level control system. This allows for easier sensor system reconfiguration, both as a means of providing greater tolerance for sensor failure, and to enhance incremental development of additional sensing and processing devices.

We have expanded the concept of Logical Sensors to also include Logical Actuators, and incorporated it into Logical Sensor/Actuator (LSA) objects. LSAs are plan execution entities, each of them corresponding to an explicit goal. Sensor Explication is basically a process of creating and manipulating these entities by activating/deactivating them and manipulating the flow of information between them. Thus, as the world changes, the entities that pro-

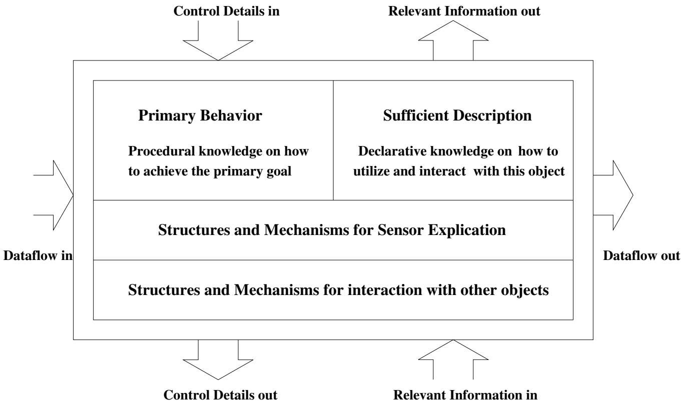

Fig. 7. The Logical Sensor/Actuator Object.

vide information and the datapaths between them change.

By defining Sensor Explication in terms of object interactions, the LSA architecture evolves from simply "plan translation into robotic code" to "coordination of intelligent plan execution objects".

For example, the high-level goal of turning the robot towards a moving object could be defined as a feedback control with little knowledge used within decomposing, determining, and selecting. The high-level goal of moving the robot through a doorway requires decomposing, determining, detecting and resolving conflicts. Sensor Explication decomposes the goal into the lower level goals of finding the doorway, moving towards it, as well as feedback control mapping of sensor-sweeping an area, searching for the doorway until found, and progressively moving through the doorway until clearing it. More details on this example are given later.

IV. System Design: Logical Sensor/Actuator Objects

We developed an object oriented architecture for Sensor Explication built around the concepts outlined in the previous section, and we have implemented it in an object-oriented programming environment. This resulted in a flexible testbed. The testbed is a collection of reconfigurable objects and flexible component structures which can be combined to achieve high-level goals through execution. A general object is shown in Figure 7.

Within the LSA object definition, there are four major sub-components: (1) Primary Behavior; (2) Sufficient Description; (3) Structures and mechanisms for explication; and (4) Structures and mechanisms for interaction with other objects. They are layered into three levels. At the lowest level are the mechanisms for interacting with other objects. The middle level contains the mechanisms for explication. These mechanisms utilize the lowest level to interact with other LSAs. The top level includes the object's Primary Behavior and Sufficient Description. The Primary Behavior defines the actions which this LSA takes, and the Sufficient Description includes, in a declarative form, the information needed by other LSAs to utilize this object.

The LSA object definition has three types of dataflow which pass through it. The first is relevant information from non-subordinate LSAs to controlling LSAs. This is the primary dataflow path where sensor data (raw or processed) pass through the LSA, and are used to set up the lowest level feedback control loops in the architecture. The second is relevant information from subordinate LSAs to a controlling LSA. This path is where internal state information passes from one LSA to another LSA in such a way to be used for control decisions. The third path is used for control details from controlling to subordinate LSAs. This is the control flow path.

V. Implementation of the Testbed

The testbed is implemented on a SUN SPARC 4/330 computer which interacts with a TRC Labmate mobile robot (nicknamed "Eric the Red") over two separate serial lines. The system is comprised of 33K of documented lines of code written in C, Lucid Common Lisp with the Common Lisp Object System and LispView windowing system.

The architecture is implemented as a hierarchy of object classes, as shown in Figure 8. The upper part of the hierarchy (not shown) defines the components that are used in the definition of a general LSA object. The lower part of the object class hierarchy (shown) defines sub-classes organized by overall LSA behavior. The leaf sub-classes (grouped into boxes) correspond to the actual classes used during the system execution. The individual LSAs in the examples shown later resulted from making instantiations of these non-generic object classes. The six primary sub-classes of LSA are explained next.

A. Six Sub-Classes of Logical Sensor/Actuator Objects

The LSA class has been divided into six sub-classes, five major and one used as a special collection point for information about entities in the robot's environment. The primary difference between the sub-classes is in how the Sensor Explication process is implemented and how the dataflow is handled.

L-Controller. The class of L-Controller represents the Sensor Explication process in its entirety, where the Primary Behavior method actually contains the Sensor Explication algorithms. The L-Controller runs the Sensor Explication process and utilizes other LSAs to accomplish its goal. Each instance of L-Controller has a different goal and thus each contains a different collection of domain knowledge that is used by the Sensor Explication process. Internal knowledge is divided into: (1) how to control the subordinate LSAs given the goal and the current situation (input to the L-Controller from its subordinates); and (2) what LSA's to select to satisfy a Relevant-Information-Need. An internal mapping is maintained between the Relevant-Information-Needs identified internally and the external LSA's which provide input for each need. Since LSAs can be built hierarchically, L-Controllers act as building blocks. The subordinate LSAs of an L-Controller can be any of the six sub-classes of LSA, including other L-Controllers.

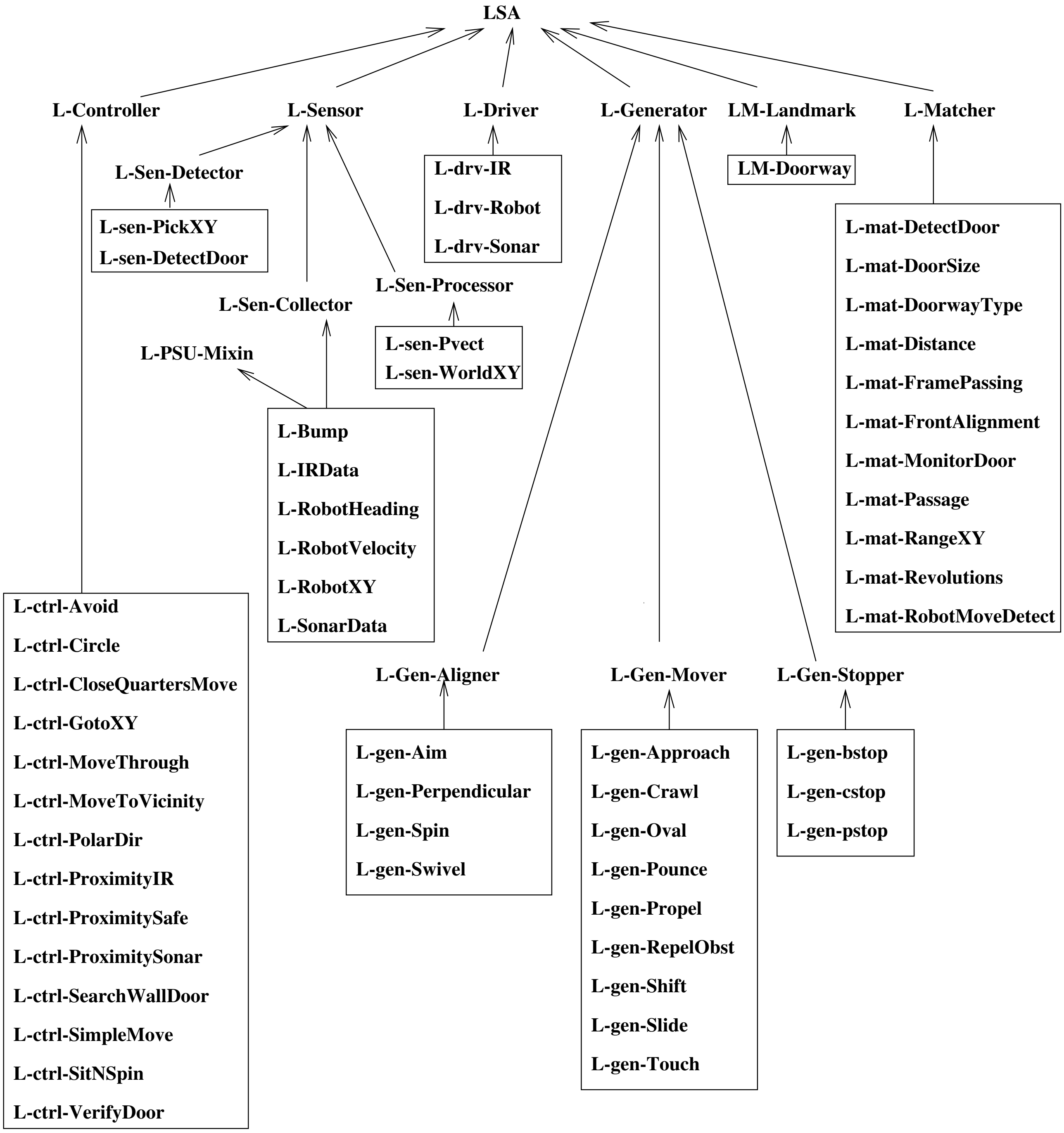

Fig. 8. The LSA object inheritance hierarchy. Arrows point to superclasses, boxes contain primitive LSAs.

L-Sensor. This is an object which produces "data" based on sensing. It can be raw data from actual hardware or it can be the result of combining the hardware with software to produce processed data. This entity is treated like a sensor, its output treated like sensor data (primarily as dataflow to satisfy another LSA's Relevant-Information-Need). L-Sensor is currently divided into three sub-classes for: (1) ordinary collection/processing of raw data; (2) classification of entity types; and (3) detection of world entities.

L-Driver. L-Driver accepts as input multiple commands for individual hardware drivers. It acts as an interface to the hardware, and performs command scheduling. It also resolves minor command conflicts, and routes major conflicts to its controlling LSA. L-Driver differs from the other classes in that its dataflow input is robot actuator commands. We use only one L-Driver per actuator. This allows the L-Driver to monitor the commands passing through it and attempt to detect conflicts, or perform command fusion and mediation.

L-Generator. The L-Generator class accepts sensor data as input and outputs a command meant for a L-Driver. This class can be viewed as a low level, feedback control, looping mechanism between a sensor and the ac-

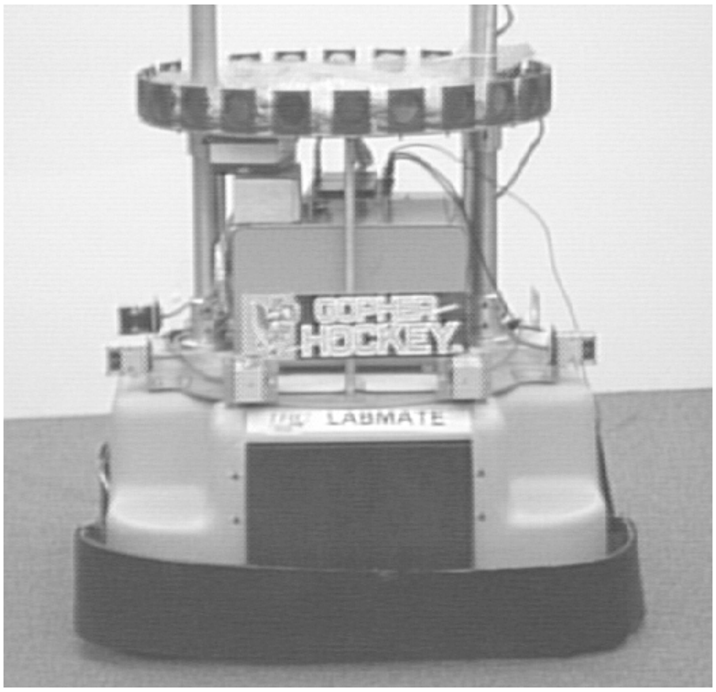

Fig. 9. Eric the Red

tuator. The Primary Behavior for most L-Generators is simply a mapping from a set of possible input values to the desired actuator command.

L-Matcher. L-Matcher is much like L-Sensor, only it also takes as control input a description of a goal or error situation. The class of L-Matcher represents the Sensor Explication sub-process of monitoring relevant information to detect goal accomplishment and error occurrence. L-Matchers use their knowledge of a particular goal or error to interpret incoming sensor data (most likely from a L-Sensor) as to whether that goal or error has occurred.

LM-Landmark. LM-Landmark is used to represent collections of knowledge about physical entities in the robot's environment. It contains information on the expected characteristics of the real entity such as expected location in the world, size, etc. Any information which might be needed by the system or which is derivable about this entity by other LSAs can be contained in this object. For example, when a LSA for detecting doors detects a specific doorway, the LM-Landmark object corresponding to the physical doorway can be updated with the last known location of the doorway. When the system, later, needs the location of the doorway and the door detection LSA is unable to detect it, the corresponding LM-Landmark object can be accessed for an estimated location to be used until the doorway detection LSA can find again the doorway.

VI. The Robot

Eric the Red is an indoor, battery powered platform, which can carry up to 200 pounds of payload (see Figure 9). The base is 70 cm wide, 75 cm long, and 28 cm high. We have outfitted the base with an ultrasonic sonar ring, an infra-red proximity ring, and an aluminum structure, which fits on top of the base, to house the sensor systems.

Twenty-four ultrasonic sensors are mounted at intervals of 15° around a 56 cm diameter ring that is 60 cm above the floor. Each of them emits an ultrasonic pulse whose cone has an angle of approximatively 30°. This arrangement provides an overlapped field coverage of the sonar beams.

Eight infra-red sensors are mounted on the robot's lower corners and can be used to determine the alignment of the robot's physical edges to objects. These sensors are most useful in determining the robot's alignment to doorways.

Eric the Red has back and front bumpers and can provide status information on the current driving mode, heading, x and y location relative to the last reset, left and right wheel velocities and accelerations, bumper contact, and motor fuses. The robot is controlled via a serial connection to a host computer which sends the robot commands and receives status information.

The shape and size of Eric the Red and the limitations of its sensors make the task of passing through doorways challenging. The majority of related research in which a robot freely passed through doorways had the luxury of wide doorways when compared to the width of the robot (often multiples of the robot's width) and, often, of using circular robots that have an easier time maneuvering through a doorway [15]..

VII. An Example of Sensor Explication

To illustrate our approach, we will use an example derived from our lab experiments [16]. When presented with the goal, "MoveThrough Door1", the system first determines what information is initially needed. The goal contains a movement command so the LSA MoveThrough is selected. MoveThrough coordinates among the three main phases illustrated in Figure 10.

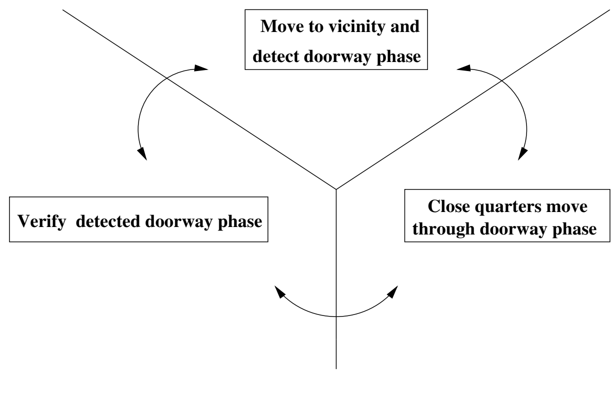

Fig. 10. The knowledge of how to move through a door guides the accomplishment of the goal by coordinating the different phases.

Details on the process are shown in Figures 11, 12, and 13. In the figures the letters at the front of each LSA oval correspond to the first letter of the subclass of that LSA.

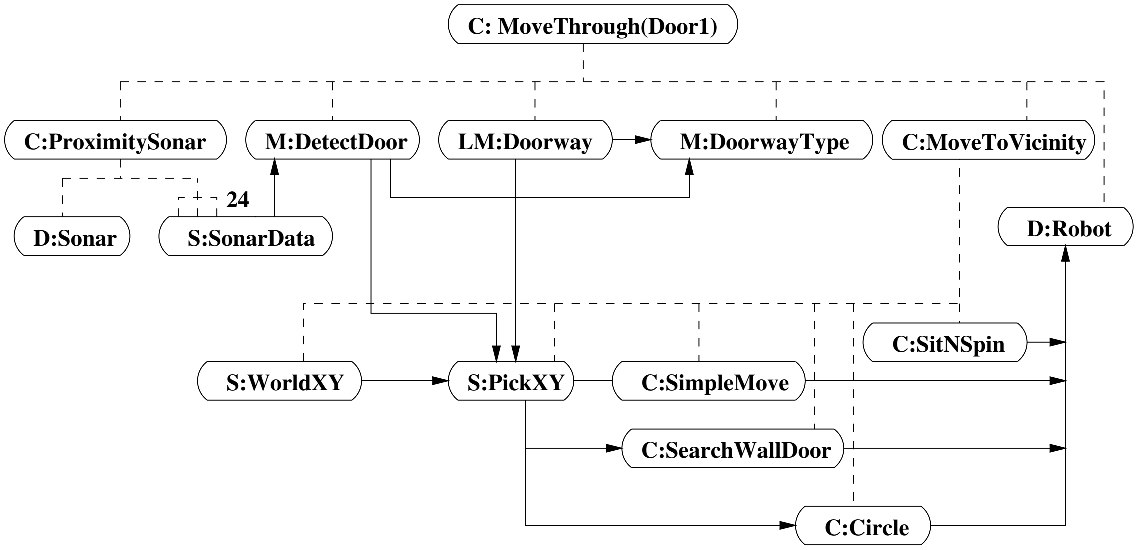

Fig. 11. First phase of moving through a door: move to vicinity and detect doorway. Continuous lines with arrows indicate the primary dataflow, dashed lines indicate the control flow.

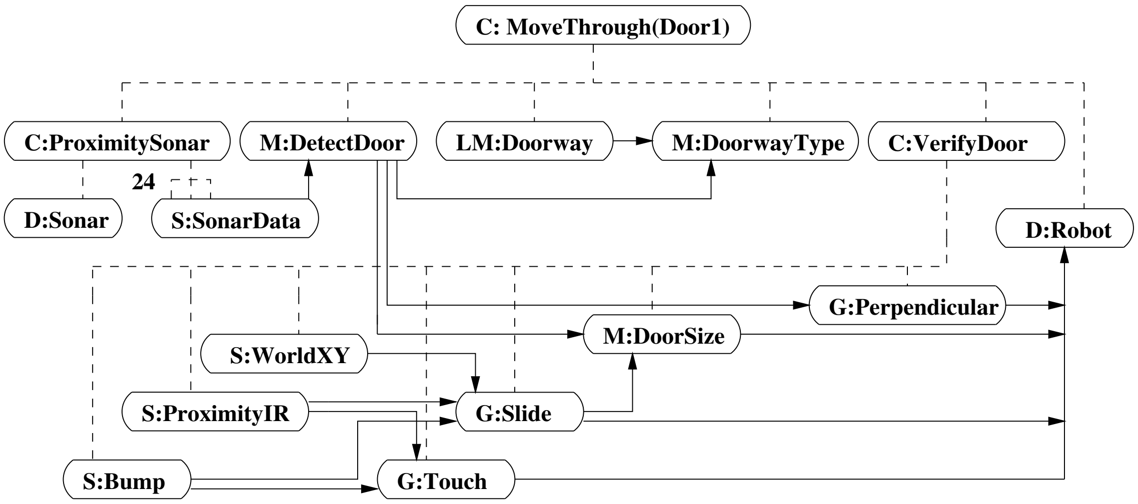

Fig. 12. Second phase of moving through a door: verify detected doorway.

A. Move to vicinity and detect doorway phase.

The initial goal requires detecting an object, Door1. Each world object has an approximate vicinity for searching for it, so the robot is required to approach that vicinity and search for Door1. Thus a decomposition is selected which includes:

1. a controller for the sonar sensors, ProximitySonar and a driver for the robot Robot. ProximitySonar spawns 24 SonarData LSAs, one for each of the available sonars, and a driver Sonar to coordinate their activity. The sonar data provide input data for other LSAs.
2. a matcher DetectDoor, that attempts to detect a doorway and, upon detection, returns its location relative to the robot. The output of DetectDoor is piped to DoorwayType. If the type of door detected is different than expected, further investigation would be needed to determine if the doorway found is the right one and how the door is positioned in it.
3. a landmark object Doorway for Door1 and a matcher DoorwayType. The doorway type is important in the selection of other LSAs and their control parameters. Doors that open in vs doors that open out appear differently, and doors in corners require different strategies.
4. a controller MoveToVicinity, that moves the robot into a designated vicinity of the doorway, using lower level LSAs. Among the LSAs that are activated by MoveToVicinity are: WorldXY, which passes the robot position to other LSAs, PickXY, which picks the likely location for the doorway, SearchWallDoor. which searches the wall looking for doors. SimpleMove, which moves the robot from one position to another utilizing lower level LSAs; SitNSpin, which

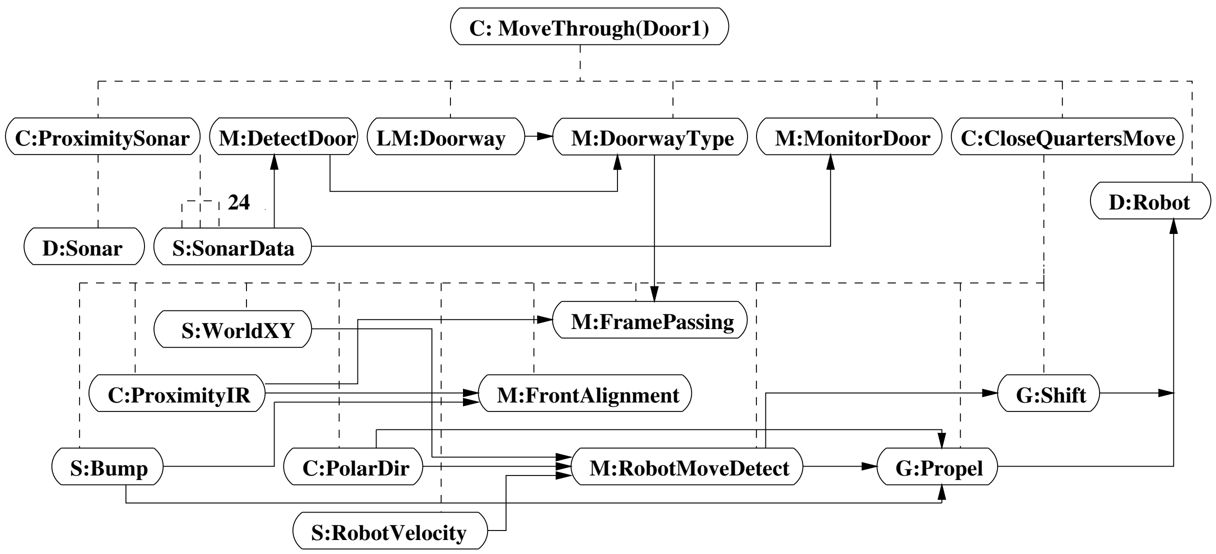

Fig. 13. Third phase of moving through a door: close quarters move through doorway.

spins the robot around in one place; CIRCLE, which circles the robot around a designated position at a designated radius.

Once a door-like structure is found, MOVETOVICINITY and its sub-LSAs are deactivated.

B. *Verify detected doorway phase.*

In the second phase, VERIFYDOOR is activated to make sure that the doorway is truly a doorway, it is open, and the robot could fit through it. This initially sets up DOORSIZE to monitor the size calculations from other LSAs to determine how much clearance there is for the robot to pass through. If the clearance is large, the robot would engage simple, yet imprecise, LSAs for propelling itself through the doorway. If it is a close fit, additional investigation must occur to verify the size and to better align the robot to the opening. The process of verifying the doorway involves three sub-phases:

1. move the robot next to the doorway, within the range of its proximity sensors (TOUCH and PERPENDICULAR),
2. move the robot back and forth in front of the opening to verify its existence and size (SLIDE),
3. center the robot to the doorway (SLIDE and PERPENDICULAR).

Both proximity and sonar sensors are used. Based on the movements within SLIDE, a more precise measurement of the doorway size is obtained, and the position of the door frame determined. This is then used to line the robot up to the doorway.

It is important to note that the switch to VERIFYDOOR is completely controlled by MOVETHROUGH. Within this LSA, the execution of the lower LSAs is monitored and upon completion of that task phase, actions are taken by MOVETHROUGH to switch to, and monitor the next phase.

C. *Close quarters move through doorway phase.*

Once the door frame is verified (assuming the passage is a close fit), VERIFYDOOR is deactivated and two new LSAs are activated. The first is MONITORDOOR, which uses the sonar sensors to monitor the robot's placement and movement through the doorway. It also detects any new obstacles that may appear on the other side of the passage.

This serves as a check on the progress of the second new LSA, CLOSEQUARTERSMOVE, which guides the robot through the opening. Upon activation, CLOSEQUARTERSMOVE also activates and controls a number of sub-LSAs that accomplish these sub-goals:

1. propel the robot through the doorway to a predicted "other side" (PROPEL);
2. keep the robot in line with the door opening as it moves through the doorway (FRONTALIGNMENT), and monitor for obstacles in the forward path using BUMP and PROXIMITYIR;
3. monitor the detection of the doorway frame (FRAMEPASSING) passing each of the side-mounted proximity sensors (PROXIMITYIR);
4. resolve any mis-alignment of the robot to the door frame (SHIFT);
5. monitor the movement of the robot for differentiating between pauses (used by SHIFT and PROPEL) and low velocity stagnation (ROBOTMOVEDETECT). Often during low velocities, the wheel motor drives will freeze up and require a reset of the velocity register.

As the robot moves through the doorway, PROPEL and FRAMEPASSING monitor for evidence of success. Once enough evidence is accumulated, CLOSEQUARTERSMOVE declares success and this prompts MOVETHROUGH to also declare the successful achievement of its goal.

VIII. Variations, Knowledge, Strategies

The above example is merely a template for what could happen and shows the anticipated sequence of actions for a typical execution. It assumes that the robot had detected the doorway right away, and that the estimated size of the doorway was not too small nor too large. In real life, each execution could yield a different sequence of actions. These variations can be produced by:

Doorsize. Though an expected doorsize is recorded in the Doorway landmark, it is merely used as a reference for what door detection parameters to use. If the actual doorsize differs greatly, the initial settings of the door detection parameters would do a poor job of detection. Thus, knowledge within MoveThrough would come into play when the robot is in the expected vicinity of the doorway and none is being detected. It then hypothesizes different sized doors and tests if any sizes match the incoming sensor data.

Door placement. The robot is initially given an expected placement of the doorway via the Doorway landmark. The actual location varies each run by as much as 1.5m. The strategy is to bring the robot to the expected location and search for the doorway. If it is not initially found, the door detection parameters are "loosened up" and various maneuvers are employed to aid in positioning the robot to detect the doorway.

Door Type and State. Another environmental unknown is the type and state of the door. The type of door is how the door opens (e.g. if there is a door, it opens in/out, to right/left) and how the physical structures around the door encompass it (e.g. is the door at the end of a hallway). The type has a great effect on how the robot would attempt to achieve the goal. In the experiments, we tested only doors of type "straight" and "wall-out-right" . The state of the door indicates whether the door is open or closed.

Ghost Doors. One of the environment interactions which often occur is the temporary appearance of a "ghost door" or the false detection of a doorway which does not exist. Initially, the higher level LSAs perform filtering via pausing the movement of the robot to see if the doorway disappears after a few sensor readings. Sometimes, the structure of the environment will cause fairly persistent ghost doors, and they are not truly discovered until the robot enters the verify detected doorway phase of the execution. Such ghosts can occur in room corners from certain angles, along the cabinets in the room (see Figures 2, 4, and 6), and at times in the middle of the room for no apparent reason (possibly the communication cable to the robot, which hangs from the ceiling, will be detected and appear as a door or obstacle).

Obstacles. Knowledge on how to avoid obstacles is basically included in two LSAs. For movement over large areas, we have an LSA that detects obstacles with the ultrasonic sensors and provides motion commands to steer the robot away from the obstacles. For close obstacles, there is another LSA that uses bumpers and infrared sensors. Upon detection, it takes control of the robot and attempts to move it away of the invading obstacle. This LSA is not used during maneuvers such as traversing through the doorway, but only in situations when objects in close proximity to the robot are not expected.

Unexpected Situations. Some of the knowledge deals with expectations. For example, if a door is detected, the robot should be able to turn towards it and move towards it until either the bumper touches one frame or the other, or the front of the robot ends up between the two frames. When the robot is commanded to "touch" the doorway, and it moves farther than expected and nothing is "touched" then this is an unexpected situation, and the system concludes that there was no real doorway. Often, when expected results are not obtained, the knowledge makes the robot fall back to a previous state where it can continue on with the achievement of its goal.

Sensor Noise. A number of strategies can be taken to reduce sensor noise. One strategy we use is to filter out the noise over a sequence of readings. Filtering works well when the majority of the values are statistically close to the true value, but that may not always be the case. If the robot happens to be sitting in a dead spot (i.e. a point where the sonar echo does not return directly back at the sensor's receiver and is thus undetectable), or an echo point (i.e. a point where multiple returns occur due to echoing) then most to all of the readings are statistically far from the true value. Filtering will not yield an acceptable value. Another problem is caused by moving objects passing near the robot that produce temporary disturbances.

Ways to handle these problems are encoded as domain knowledge within the LSAs. The robot uses its knowledge to decide what to monitor and, upon detection, how to respond with the best strategy.

IX. The Experiments

The laboratory experiments were conducted in a typical indoor open-office environment, simulating an office building with cubicles, for instance, or a warehouse for delivering parts. The environment contains furniture, allows human traffic through it, and hosts multiple doors of various sizes (see Figures 2, 4, and 6). The ceiling is constructed of a sound absorbing material, and neither it nor the floor cause specular noise to the ultrasonic sensors. The walls readily reflect both the ultrasonic and infra-red beams. The furniture reflects ultrasonic beams, but not infra-red. The laboratory floor is carpeted which results in wheel slippage and sometimes noise. Boxes were often used to create makeshift doorways of similar size to the laboratory's doorways. This avoided potential damage to the robot due to collisions. The boxes were primarily white and readily reflected both the ultrasonic and infra-red beams.

The laboratory experiments consisted of three parts: (1) development of the initial implementation on a real robot; (2) knowledge acquisition and refinement through *guided*

TABLE I
Results of capabilities experiments obtained with five runs for each test set.

Test Set	Door Type	Door Size	Obstacles	Doorway	#goal	#fail	%goal
1	straight	100 cm	none	wide	4	1	80%
2	straight	100 cm	none	wide	4	1	80%
3	straight	81 cm	none	average	5	0	100%
4	straight	81 cm	none	average	4	1	80%
5	straight	70 cm	none	too small	4	1	80%
6	straight	60 cm	none	too small	5	0	100%
7	wall-out-right	81 cm	none	actual	3	2	60%
8	wall-out-right	81 cm	none	actual	4	1	80%
9	wall-out-right	81 cm	2	actual	3	2	60%
10	wall-out-right	81 cm	2	actual	3	2	60%
Totals	2 door types	4 sizes	none or 2	50 runs	39	11	78%

TABLE II
Summary of experiments.

Characteristics	No. of runs	No. of goal	% goal	Error		
				false-goal	intervention	timeout
straight door	30	26	86.7%	2	1	1
wall-out-right	20	13	65%	0	7	0
not too small	40	30	75%	1	8	1
too small	10	9	90%	1	0	0
no obstacles	40	33	82.5%	0	4	1
obstacles	10	6	60%	0	4	0
Total	50	39	78%	2	8	1

experiments; and (3) final *capabilities experiments*.

The guided experiments were devised to improve the knowledge we originally embedded into the system. Over forty guided runs were executed. New knowledge was encoded back into the system. For example, if during the guided experimentation, it was determined that the robot was prone to become misaligned with the doorway passage upon entering it, additional knowledge (and the necessary supporting LSAs) would be introduced to monitor for the occurrence of misalignment and to correct for it.

To validate the robustness of the approach when faced with previously unencountered situations, we did not use in the guided experiments all the conditions we expected to face later in the capabilities experiments. None of the guided experiments were conducted with doors too small, nor with the door type "wall-out-right". Door sizes ranged from 81 cm to 95 cm and obstacles rarely came into play.

The primary difference between the guided and the capabilities experiments is that during the capabilities experiments no change of the code was allowed and all experiments had to run to completion.

Completion is defined as the robot entering one of these states:

goal: the robot successfully and knowingly passes through the doorway;

false-goal: the robot fails by declaring it has passed through the doorway when it actually has not;

intervention: The robot fails due to human intervention to avoid damage (e.g. the robot runs into an obstacle, but does not detect it and continues to push into it);

timeout: the robot's execution has surpassed a time limit set for the task.

We could think of two other states of completion, in which the robot fails in passing through the doorway, but acknowledges the failure, or passes through the doorway without knowing it succeeded. Actually, the first is never a final experiment result, because the robot would realize that it is failing and initiate a maneuver which brings it back near its starting point to start all over. The robot will try, try again, until one of the other results occurs. The second happened once in the guided experiments but never in the capabilities experiments.

A total of 50 capabilities experiments were run with 10 different scenarios of five runs each. The scenarios varied in distance traveled to goal, doorway sizes, doorway types, and obstacles. Test sets 1 through 6 were with a doorway constructed of cardboard boxes, which allowed changes in its position and size. Sets 7 through 10 were with a real door that opens out and to the right.

Of the 50 capabilities experimental runs, 78% were completed successfully. The results are shown in Table I. The results from the experiment runs which did not have obstacles were surprisingly good. The robot succeeded 90% of the time in determining that the doorway was too small, which is fairly good considering the lack of guided experiments in this area. The same can be said for the results of the "wall-out-right" runs, where walls are to the immediate right of the doorway. The clearance between the robot and the door frame in Test Sets 7 through 10 is very limited, and requires precision in sensing and control beyond the individual sensors available on our robot. The experiments provide evidence of the robustness of the approach and of transferability of knowledge to never-seen-before scenarios.

In the eleven cases that did not succeed, as shown in Table II, eight were caused by the robot getting wedged in the doorway (seven of those where with the wall-out-right door type), two were caused by the robot declaring it had successfully traversed the doorway when it fact it did not, and one was caused by the robot working itself into a corner and never escaping (and so failing because of timeout). A detailed analysis of the experiments is in [17].

The robot's performance in dealing with ghost doors was extremely good. The door detection LSA often detected ghost doors in the middle of the room or on doorway-less walls. The robot would approach the ghost door and start its verification procedures and always determined it to be a false door.

One unexpected observation was that the robot had problems maneuvering near the edges of the doorway. If the robot ever became misaligned with the doorway so that it was facing towards one frame with the other frame against its side, it would be in serious trouble. Most of the error-intervention runs resulted from this situation. The development of specific domain knowledge and a more sophisticated proximity detection method could help in this situation.

Another unexpected observation was that the robot had difficulty with the widest doorway (100 cm). The door detection LSA could not quickly nor consistently detect the wide opening. This became especially true at very close ranges, such as at the points in the Verify phase where the robot would attempt to become perpendicular/parallel to the opening. At that close range, DetectDoor would loose the tracking of the door and eventually determine that it must have been a ghost door. Upon moving back to the middle of the room, the robot would again detect the doorway and approach to verify. It would take a number of repeated tries at this until, by chance, the verify phase occurred far enough away from the opening to allow DetectDoor to continue tracking it. Persistence does pay off, and the knowledge encoded within MoveThrough performed well at re-attempting to verify what appeared to be the correct door. This is a good example of proper application of knowledge to improve overall performance.

X. The Use of Knowledge

The role or importance of the domain knowledge is extensive. The questions are: what knowledge is encoded? how is it encoded? how was it acquired? and what are its limitations? These issues are discussed next.

Domain Dependent vs. Domain Independent Knowledge. One of the goals of this research was to be able to separate the domain dependent knowledge from the domain independent knowledge. The Sensor Explication process implemented in the L-Controller class methods is domain independent, while the knowledge encoded in rules that is used by Sensor Explication is primarily domain dependent. The advantage of this division is that it separates the Sensor Explication process from the rest of the computation. This simplifies extending the architecture to other domains, tasks, and environments.

Proper Application of Knowledge Increases Performance. Failure is mostly due to limitations and weaknesses in the LSA's knowledge. Many LSAs have simple algorithms. Poorly designed LSAs resulted in a higher need for knowledge to compensate for the lack of performance. A good example is ProximitySafe. Its poor performance is evident by the amount of difficulty the robot experiences in moving away from obstacles. One of the capabilities experiments failed due to its inability to retreat. Another example is DetectDoor. The algorithm had troubles in detecting larger doorways. If not for the knowledge encoded in its controlling LSA, this weakness could have caused additional failures. The knowledge in the LSA dealt with monitoring for large doorways and applied various maneuvering strategies to facilitate the detection of the doorway as problems occurred. Because of this, the process of detecting and tracking the doorway through the entire execution of the goal succeeded, though progress was occasionally hampered by noise.

This truly emphasizes the importance for higher levels of control to have knowledge about the weakness and limitations of the lower levels. Areas containing knowledge performed very well compared to those lacking knowledge. The laboratory experiments showed that even simple algorithms can be combined with knowledge into an overall synergistic algorithm which is far more reliable than any of the individual components.

Knowledge Acquisition Through Experimentation. The success of the laboratory experiments showed that knowledge can be extracted from the domain and implemented in the system. The experiments also showed the difficulty in extracting the knowledge and structuring it to be properly utilized. At times a great deal of investigation would be needed to determine what aspect of the environment was causing a failure of a LSA or a poor coordination within a dataflow. Knowledge acquisition was the largest bottleneck during the experiments.

Limits of Domain Knowledge. The primary problem is the dependence of the system on the domain knowledge. Though the building of the domain knowledge seems straightforward, there is still a kind of "black magic" aspect about it. The main problem that has plagued domain knowledge all along still remains. There must be domain knowledge for each determined scenario or discovered weakness. The resulting system is only as powerful as the scenarios the knowledge covers, and there is no guarantee that the knowledge encoded will work beyond the scenarios examined during knowledge acquisition. Thus, the amount of domain knowledge can be endless, and the question remains of "how much is enough". We have observed, as expected, that initially each increment of knowledge improved the performance greatly. However, as the total amount of knowledge increased, the performance did not increase by a comparable amount. The returns were diminishing. At some point the performance improvement is no longer worth the cost. The issue then becomes one of desired performance vs. cost. The issue can only be answered within each individual application domain (i.e. how many dollars is robustness worth to this application?).

XI. Related Research

There has been a large amount of research in the area of sensor-based control of a mobile robot. The most popular and successful approach is the subsumption architecture [8] which avoids planning altogether by using layers of behaviors. Behaviors run in parallel and interact when needed. Each individual layer corresponds to a level of behavioral competence. Though this method of control allows for many behaviors, it disallows central control and so it makes it difficult to assign to the robot different goals to be achieved.

Our method of "intelligent plan execution objects" blends well into the "behaviors" paradigm which dominates the autonomous robotics research today. Using LSAs, a behavior can be constructed from a set of objects, and controlled by a single object which corresponds to that behavior. Each of the sub-objects can attend to a different aspect of that behavior, such as producing information from the sensor data, or deciding what distance to move or turn, or deciding when the goal has been achieved and it is time to stop. In this sense our work is similar to the Perception and Motor Schemas by Arkin [9]. The main difference is in the way LSAa are activated. Control in Arkin's approach is achieved by a single control module that selects from a single layer of motor schemas. In our system knowledge stored in LSAs controls how LSAs are combined together. Different arbitration and task decomposition methods can be programmed in the LSAs, providing for more flexibility.

Many architectures have been proposed that combine the ability to react to the environment with planning. Georgeff [18] has implemented a planning and control system based on a functional decomposition of high-level control into primitives, and that integrates some reactive behaviors into the control structure. The Rex/Gapps system [19] decomposes high-level goals into mobile robot commands, and then converts them into hardware logic designs. The result is a large reactive system that has goal-oriented behavior but cannot plan or adapt itself to new goals.

The 3T architecture [20] has three levels: deliberative, sequencing, and reacting. The top level performs activities such as planning and world modeling, the middle level draws directly from the RAP system [21], while the lower level is a stateless reactive control mechanism which controls activities with no decision-making.

Firby's most recent work [1] extends his original RAP architecture by allowing the creation of independent reactive agents for different tasks, and the run-time selection of which one to use to achieve the next subtask. His "skills" correspond to our low level LSAa, and "skill networks" to our high-level LSAs. The primary difference is that our architecture is homogeneous. Everything is a Logical Sensor/Actuator, from the low-level drivers for sensors and actuators to the high-level LSAs for abstract goals. Common methods are shared by objects through inheritance. The ability to activate/deactivate objects and to change the dataflow among them in a dynamic way depending on the situation gives us a flexibility that other architectures do not have.

The Task Control Architecture (TCA) of Simmons [3] is another successful example of combining deliberative and reactive control. The TCA consists of a set of task-specific modules and of a central control module that coordinates their activities. The central control module routes messages dynamically to the other modules. The TCA supports the creation and execution of hierarchical plans, and facilitates incremental building of systems. The LSA architecture is more distributed and has no central controller.

A large number of experimental results obtained with various robots and tasks using the architectures described above and others not mentioned attests to the validity of developing flexible and reconfigurable architectures.

The LSA architecture borrows heavily from the idea of Logical Sensors [14], [22]. Logical Sensors are abstracted views of sensors and sensor processing, much like how logical I/O is used to insulate the user from the differences of I/O devices and operating systems. We have expanded the concept of Logical Sensors to include Logical Actuators, incorporated it into a comprehensive framework, and demonstrated experimentally the viability of the idea

There are also similarities between LSA and blackboard architectures [4]. Both hierarchically organize different components. The blackboard agent architecture's components are: the perception processes (similar in functionality to L-Sensors); the action systems (similar to L-Drivers); the reflex-arcs and perception-action coordination processes (similar to L-Generators) and the cognition system (similar to L-Controllers and L-Matchers). The main differences hinge on centralized vs distributed control and data storage. The blackboard architecture has a single cognition system containing multiple knowledge sources. The LSA distributes the control across multiple L-Controllers. The storage of data is centralized on the blackboard, while the LSA architecture requires data produced by a LSA to be stored within that LSA.

The CIRCA system [5] integrates planning with real-time control. Most of the work has focused on generating plans in which the control actions are guaranteed to be executed within the real-time constraints. Our research has not addressed the issue of how to guarantee real-time performance. Our architecture separates planning from execution, and simplifies planning by increasing the level of intelligence of the execution system. The knowledge encoded in LSAs enables the execution system to be flexible and capable of taking advantage of the resources and situations available in achieving its goal.

XII. Conclusions

The Logical Sensor/Actuator architecture was developed as an approach to robotic execution of classical planning goals. This architecture addresses the need to handle noisy sensors and actuators, the need to remain reactive to changes in the environment while accomplishing goals, and, most important, the application of knowledge to deal with resolving failures.

The LSA architecture provides evidence that a goal can be achieved through proper application of knowledge. Specifically, this entails identifying (1) what information is relevant to achieving the goal; and (2) what are the sources of this information (i.e. sensors, actuators and processes), and how to control them. Extensive lab experiments demonstrate that the architecture is reliable and robust, despite sensor noise, actuator errors, and even limits in the knowledge encoded in the LSA objects.

Acknowledgments

We would like to thank Scott Brandt, John Fischer, Chris Smith, Karen Sutherland, and Eric the Red for their help, and the anonymous reviewers for their suggestions that greatly improved this paper. All research described here was performed at the Artificial Intelligence, Robotics and Vision Laboratory of the Department of Computer Science, University of Minnesota.

References

[1] R. James Firby, "Task networks for controlling continuous actions," in *Proc. Artificial Intelligence Planning Systems*, 1994.

[2] D. Lyons, "Representing and analyzing action plans as networks of concurrent processes," *IEEE Transactions on Robotics and Automation*, vol. RA-9, no. 3, pp. 241–256, June 1993.

[3] R. Simmons, "Structured control for autonomous robots," *IEEE Transactions on Robotics and Automation*, vol. 10, no. 1, pp. 34–43, February 1994.

[4] Barbara Hayes-Roth, "An architecture for adaptive intelligent systems," *Artificial Intelligence*, vol. 72, no. 1-2, pp. 329–365, 1995.

[5] David Musliner, Edmund Durfee, and Kang Shin, "World modeling for the dynamic construction of real-time control plans," *Artificial Intelligence*, vol. 74, no. 1, pp. 83–127, 1995.

[6] D. E Wilkins, K. L. Myers, J. D. Lowrance, and L. P. Wesley, "Planning and reacting in uncertain and dynamic environments," *Journal of Experimental and Theoretical Artificial Intelligence*, vol. 6, pp. 197–227, 1994.

[7] J. S. Albus, "Hierarchical interaction between sensory processing and world modeling in intelligent systems," in *5th IEEE Symp. on Intelligent Control*, 1990, pp. 53–59.

[8] Rodney Brooks, "A robust layered control system for a mobile robot," *IEEE Journal of Robotics and Automation*, vol. RA-2, no. 1, pp. 14–23, March 1986.

[9] R. C. Arkin, "Motor schema-based robot navigation," *International Journal of Robotics Research*, vol. 8, no. 4, pp. 92–112, Aug. 1989.

[10] D. W. Payton, J. Kenneth Rosenblatt, and David M. Keirsey, "Plan guided reaction," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 20, no. 6, pp. 1370–1382, 1990.

[11] E. Gat, "Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots," in *Proc. Nat'l Conf. on Artificial Intelligence*, 1992, pp. 809–815.

[12] J. Budenske and M. Gini, "Why is it so difficult for a robot to pass through a doorway using ultrasonic sensors?," in *IEEE International Conference on Robotics and Automation*, 1994, pp. 3124–3129.

[13] E. Gat, "Robot navigation by conditional sequencing," in *IEEE International Conference on Robotics and Automation*, 1994, pp. 1293–1299.

[14] T. Henderson and E. Shilcrat, "Logical sensor systems," *Journal of Robotics*, vol. 1, no. 2, pp. 169–193, 1984.

[15] E. Gat, "Robust low-computation sensor-driven control for task-directed navigation," in *IEEE International Conference on Robotics and Automation*, 1991, pp. 2484–2489.

[16] John Budenske and Maria Gini, "Knowledge, execution, and what sufficiently describes sensors and actuators," in *SPIE OE/Technology '92*, 1992.

[17] John Budenske, *Robotic Plan Execution in Dynamic and Unpredictable Environments*, Ph.D. thesis, University of Minnesota, 1993.

[18] Michael P. Georgeff and Amy L. Lansky, "Reactive reasoning and planning," in *Proc. Nat'l Conf. on Artificial Intelligence*, Seattle, WA, 1987, pp. 677–682.

[19] Leslie P. Kaelbling and Stan Rosenschein, "Acting and planning in embedded agents," *Robotics and Autonomous Systems*, vol. 6, pp. 35–48, 1990.

[20] R. P. Bonasso, H. Antoinisse, and M. G. Slack, "A reactive robot system to find and fetch tasks in an outdoor environment," in *Proc. Nat'l Conf. on Artificial Intelligence*, 1992.

[21] R. James Firby, "An investigation into reactive planning in complex domains," in *Proc. Nat'l Conf. on Artificial Intelligence*, Seattle, 1987, pp. 202–206.

[22] T. Henderson, C. Hansen, and B. Bhanu, "The specification of distributed sensing and control," *Journal of Robotic Systems*, vol. 2, no. 4, pp. 387–396, April 1985.

John Budenske received his B.S., M.S, and Ph.D. degrees in Computer Science from the University of Minnesota in 1983, 1987, and 1993, respectively. He worked at the Honeywell Systems and Research Center on artificial intelligence, signal image processing, robotics, simulation, and database systems. He then worked on advanced methods for software reengineering and object oriented programming at Loral Defense Systems. Currently, he is performing research in intelligent control, software engineering, electronic collaboration, and intelligent agents for Architecture Technology Corporation in Minneapolis, Minnesota.

Maria Gini is an Associate Professor in the Department of Computer Science at the University of Minnesota, in Minneapolis. She has been a Research Associate in the School of Engineering, Politecnico of Milan, Italy and a Visiting Research Associate at the Artificial Intelligence Laboratory at Stanford University. Her research interests are in integrating Artificial Intelligence with robotics for navigation, exploration, learning of mobile robots, parallel algorithms for real-time path planning for articulated robots, task planning with incomplete or uncertain information, detection and recovery from robot errors, object-oriented programming for robotics.