To appear in *Journal of Experimental and Theoretical Artificial Intelligence,*
Special Issue on Software Architectures for Autonomous Agents

Logical Sensor/Actuator: Knowledge-Based Robotic Plan Execution

John Budenske and Maria Gini

Department of Computer Science, University of Minnesota
4-192 EE/CSci Building, 200 Union Street SE, Minneapolis, MN 55455

move the header up from body of text (headsep) and compensate in topmargin

Abstract

Complex tasks are usually described as high-level goals, leaving out the details on how to achieve them. However, to control a robot, the task must be described in terms of primitive commands for the robot. Having the robot move itself to and through an unknown, and possibly narrow, doorway is an example of such a task. 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. We illustrate our approach using actual data from a real robot.

1 Introduction

A large part of our world is unstructured and dynamic. This fact has been one of the leading obstacles to mass-quantity introduction of robots into everyday life. The variability of the world makes it impractical to develop detailed plans of actions before execution. The environment might change and thus invalidate the plan. It makes more sense to develop, before execution, a detail-less plan of what to do and make the necessary details explicit during execution. Thus, we are faced with a problem: how can a robot achieve a high-level goal and still remain reactive to its environment? Many approaches have been proposed to make robots reactive [Brooks, 1986] and to combine planning with reactivity [Bonasso *et al.*, 1992, Firby, 1994, Simmons, 1994].

In this paper 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 knowledge about the domain and about the sensors and actuators available. Our approach is based on three premises: (1) plan execution requires many details that are hidden in the plan's abstraction to be made explicit; (2) plan execution

is an information-gathering based process, where determining what information is relevant constitutes a great deal of the process; and (3) knowledge is integral to plan execution, and thus proper application of knowledge increases the robustness of plan execution.

We describe a coherent approach that allows specification, coordination, and control of a variety of sensors and actuators in a reusable and flexible architecture, that we call *Logical Sensor/Actuator (LSA)*. The implementation of our solution resulted in an object oriented system called the *Logical Sensor/Actuator Testbed* which contains a library of reusable and reconfigurable sensor and actuator processing entities. The testbed can be extended to include other sensors or robots, re-applied to other multisensor/multiactuator systems, or converted for simulation systems.

We have focused our investigation on a particular domain within robotics where the robot is given a detail-less plan, such as "move through doorway", that has to be executed in a changing and only partially known environment. This particular application domain contains all the relevant features of the multisensor/multiactuator domains we are interested in, and was readily available to us for experimentation. Given the robot and sensors available to us, the task requires precision beyond that of any single sensor, and requires the combination of multiple sensors, so providing a rich domain. The approach we developed has a much wider range of applicability, not only across domains, but across platforms (mobile robots, manipulators, softbots, etc.) as well. Any situation where decisions on how to proceed depend on both a given goal and on the interaction with the "outside world" via multiple "information sources" would be a good application for this work.

2 System Design: Logical Sensor/Actuators Objects

To be executed by a robot, a task must explicitly specify how to utilize the sensors and actuators. For the same goal, different situations often require different sensors and different strategies. Because this transformation process depends on the current situation it must occur at execution time. Essentially, plan execution requires two steps:

identifying what information is needed for execution. Plan execution 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".

finding and controlling sources for that information. Plan execution 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 needed to "achieve X".

Abstraction is often used to reduce dependency on the source of information. We have utilized the concept of Logical Sensor, proposed originally by [Henderson and Shilcrat, 1984]. A Logical Sensor contains both a declarative description of the sensor and procedures to control it. We have expanded the concept of Logical Sensors to include Logical Actuators and embedded it into objects that we call Logical Sensor/Actuator (LSA). LSAs are plan execution entities, each of them corresponding to an explicit goal. Each LSA is capable of performing a process that we call *Sensor Explication*[1].

Sensor Explication is basically the process of creating and manipulating these entities by activating and deactivating them and manipulating the flow of information between them. Thus, as the world changes, the entities that provide information and the data paths 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".

An LSA object has three primary functionalities:

1. An object is a source of information. When an object is given a high-level goal, its Sensor Explication process needs to find the information necessary for execution. For each information need a source has to be selected. The source will then provide the information and the control values to satisfy the original object's need.

2. An object includes a knowledge-base on how to achieve the goal it is designed to achieve. This is the knowledge that the Sensor Explication process utilizes.

3. An object includes a knowledge-base about how to use itself. An object must select between, interact with, and utilize other objects as sources of information. To do so, it utilizes knowledge about the other objects. The best location for knowledge about an object is within the object itself.

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

The use of objects allows the knowledge base to be partitioned across "understandable" or "graspable" entities. For instance, the knowledge concerning how to avoid bumping into obstacles is in one object while the knowledge concerning detecting the doorway is in another. We have found that focusing our attention on entities and their knowledge content more than on functional requirements, as suggested by [Bailin, 1989], helped the design and implementation process.

[1]from the Websters dictionary – noun: (origin 1531) 1: a logical analysis such to make explicit 2: a detailed explanation of.

3 Implementation of Logical Sensor/Actuator Testbed

We have developed an object oriented architecture and implemented it in an object-oriented programming environment resulting in a flexible 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 (CLOS) and LispView windowing system. The architecture is, basically, a collection of reconfigurable components and flexible component structures which can be combined to achieve high-level goals. It contains methods for describing sensors, actuators, processes as well as abstractions of them.

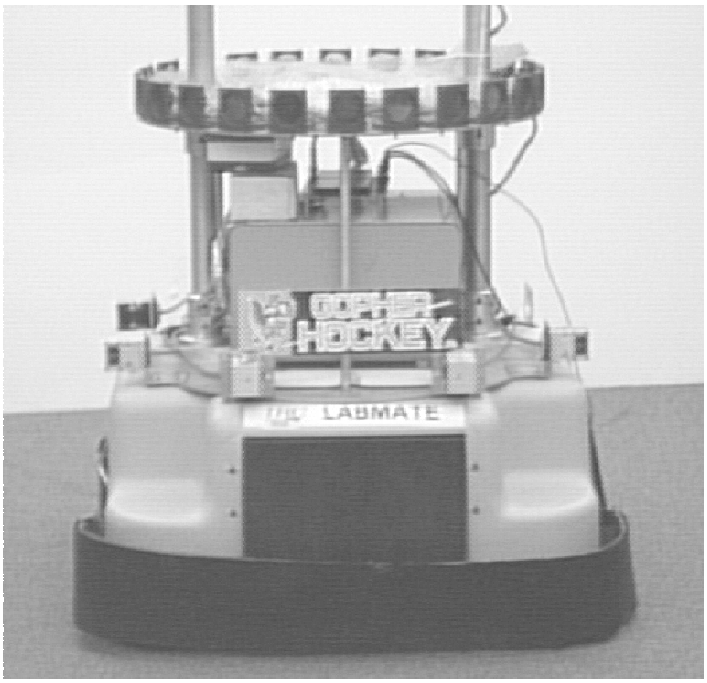

Figure 1: Eric the Red

Eric the Red is an indoor, battery powered platform with a complete control system and which can carry up to 200 pounds of payload (see Figure 1). 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 a 30° angle. 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.

The testbed is implemented as a hierarchy of object classes. The LSA class has six sub-classes. The five major sub-classes represent the different sub-functionalities within Sensor Explication. The primary difference between them is how they handle the dataflow. The sixth sub-class is a special collection point for information about entities in the robot's environment.

L-Controller. The class of L-Controller represents the Sensor Explication process in its entirety. The L-Controller runs the Sensor Explication process over its designated goal and utilizes other LSAs to accomplish the goal. Each instance of L-Controller has a different designated goal which it accomplishes and thus each contains a different collection of domain knowledge used by the Sensor Explication process. The internal knowledge is split into: (1) how to control the subordinate LSAs given the goal and the current situation. The current situation is known to the L-Controller through information passed from its subordinate LSAs; and (2) what LSAs to select to provide the needed information. An internal mapping is maintained between the information needs identified internally and the external LSAs which provide input for each need. Thus flexibility and modularity is maintained between L-Controllers and their subordinates. Since LSAs can be built hierarchically, L-Controllers act as the building blocks. The subordinate LSAs of an L-Controller can be any of the six sub-classes of LSA, including other L-Controllers.

Among the L-Controllers we have defined for the problem of finding and moving through a doorway are:

- *MoveThroughDoor*, which controls multiple configurations of lower level LSAs (e.g. such as thoses listed below) to guide the robot through the doorway;
- *VerifyDoor*, which controls multiple configurations of lower level LSAs to verify that a detected doorway is truly a passable doorway;
- *MoveToVicinity*, which moves the robot into a designated vicinity, using various configurations of lower level LSAs;
- *Avoid*, which detects and avoids obstacles as the robot is moving;
- *Circle*, which circles the robot around a designated position at a designated radius;
- *CloseQuartersMove*, which guides the robot through an opening currently detected by its infrared sensors;
- *GoToXY*, which moves the robot to a location designated by dead reckoning coordinates;
- *ProximitySafe*, which upon detection of a very close obstacle takes control of the robot to guide it away from the obstacle until *SimpleMove, GoToXY,* or *Avoid* can return to guiding without error;

- *ProximitySonar*, which controls the 24 sonar LSAs *Range-Psonar* and the sonar ring controller *ProximitySonarDriver*.
- *Sit-N-Spin*, which spins the robot around in one place;
- *SimpleMove*, which moves the robot from one position to another utilizing lower level LSAs;

L-Sensor. This is an object which produces data based on sensing. The data can be raw data from actual hardware or can be the result of combining hardware with software to produce processed data. Elements of this class are treated like sensors, their output treated like sensor data.

Among the L-Sensors we have defined are:

- *Range-Psonar*, which collects and checks data from an individual ultrasonic sensor.
- *Robot-Bump*, which monitors the sensor readings from the robot's bumpers and determines when there is truly an object touching the bumpers;
- *Robot-XY*, which monitors the robots dead reckoning sensor and determines its current X,Y position. This includes monitoring for hardware resets which will zero out the dead reckoning sensor.

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 an actuator. Most L-Generators simply map from a set of possible input values to the desired actuator command.

Among the L-Generators we have defined are:

- *Perpendicular*, which moves the robot to be aligned and perpendicular to the doorway;
- *Pounce*, which utilizes different methods to quickly move the robot;
- *Propel*, which slowly propels the robot through a small area (e.g. the doorway opening) to a predicted open area (e.g. other side of doorway);
- *Shift*, which maneuvers the robot to perform a lateral move left or right.
- *Slide*, which moves the robot back and forth in front of the doorway depending on sensor detections;
- *Touch*, which moves the robot until an object comes within the range of the proximity sensors.

L-Driver. L-Driver accepts as input multiple commands for individual hardware devices. 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-Drivers differ from the other four classes in that their input are robot actuator commands.

Among the L-Drivers we have defined are:

- *ProximitySonarDriver*, which coordinates the activity of the Range-Psonar LSAs.

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

Among the L-Matchers we have defined are:

- *DetectDoor*, which attempts to detect a doorway in the environment and upon detection returns its location relative to the robot;
- *Doorsize*, which monitors the calculated size of the doorway opening;
- *DoorwayTyper*, which determines the type of doorway based on data piped from a LSA such as *DetectDoor*;
- *FrontAlignment*, which tracks and adjusts the robots alignment with an opening via using the infrared sensors.
- *FramePassing*, which monitors the detection of the doorway-frame passing each of the side-mounted proximity sensors;
- *MonitorDoor*, which uses the sonar sensors to monitor the robot's placement and movement through the doorway. It also detects any new obstacles which may appear on the other side of the passage;

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 its 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 stored in a LM-Landmark object.

Among the LM-Landmarks we have defined are:

- *LM-Doorway(Door1)*, which collects and holds information on doorways, specifically one labeled "Door1".

4 An Example of Sensor Explication

To illustrate our approach, we will use an example derived from our lab experiments [Budenske and Gini, 1992]. When presented with the goal, "MoveThrough Door1", the system first determines what information is initially needed. This initial goal requires detecting

an object, Door1. Each world object is associated with a LM-Landmark LSA which collects and holds information on that object. If a LM-Landmark LSA exists for Door1 *LM-Doorway(Door1)*, it is activated, else one is instantiated for it. The LM-Landmark for Door1 is expected to provide a vicinity to search for Door1 (e.g. from X,Y dead reckoning locations recorded earlier, and from default rules on where to look for entities such as doors). The goal also contains a movement command so the LSA *MoveThrough* is selected. *MoveThrough* coordinates among the following three main phases:

Move to vicinity and detect the doorway. Given the need to search for and find Door1, a decomposition is selected and its datapaths are configured. This includes:

1. approaching the vicinity of the doorway using *MoveToVicinity*. *MoveToVicinity* in turn configures a dataflow of sub-LSAs to achieve its goal of moving the robot to the given vicinity and then to maneuvering around to aid the higher level goal, that, in this case, is to detect the doorway. The sub-LSAs used include *SimpleMove GoToXY*, *ProximitySafe*, *Pounce*, *Circle*, *Sit-N-Spin* , and *Avoid* .
2. searching and detecting the doorway, using *DetectDoor*, that performs the search using sonar input;
3. linking *LM-Doorway(Door1)* into the configuration along with *DoorwayTyper* for determining the type of doorway being sought. The doorway type is important in the selection of other LSAs and their control parameters. Doors which open in vs doors which open out will appear differently to many of the sensor processing LSAs and thus a prediction of the door type allows more specific processing to occur;
4. activating (*ProximitySonar*) to control the sonar sensors.

Once a door-like structure is found, *MoveToVicinity* and its sub-LSAs are deactivated. *DetectDoor* is kept to aid in monitoring the other phases. Its output is piped to *DoorwayTyper* in case the type of door detected is different than expected or initially sensed. This would lead to further investigation on whether the right doorway was actually found and how the door is positioned in it (e.g. open, close, partial).

Verify the detected doorway. In the second phase, a *VerifyDoor* LSA is activated to make sure that the doorway is truly a doorway, it is open, and the robot could fit through it. *VerifyDoor* initially sets up *DoorSize* to monitor the size calculations from other LSAs. This is needed to determine how much clearance there is for the robot to pass through. If there is a large clearance, the robot would select simple, yet imprecise, LSAs for propelling itself through the doorway. If it is a close fit, additional investigation must occur to further verify the size and to better align the robot to the opening. The later process of verifying the doorway involves utilizing both infrared proximity and sonar sensors during the following three sub-phases:

1. move the robot next to the doorway, within the range of its proximity sensors using *Touch* and *Perpendicular*;
2. move the robot back and forth in front of the opening using *Slide* to verify the existence of the opening and obtain a more precise measurement of its size as well as the position of the door frame;
3. center the robot to the doorway using *Slide* and *Perpendicular*.

It is important to note that the switch from the initial LSAs 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 the next phase.

Move through the doorway. Once the door frame is verified, *VerifyDoor* is deactivated and two new LSAs are activated, *MonitorDoor* and *CloseQuartersMove*. In this example we assume the passage is a close fit. *MonitorDoor* monitors the robot moving through the doorway, and detects any new obstacles. It also serves as a check on the progress of *CloseQuartersMove*, as it guides the robot through the opening. Upon activation, *CloseQuartersMove* 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" using *Propel*;
2. keep the robot in line with the door opening as it moves through the doorway, and monitor for obstacles in the forward path using *FrontAlignment*;
3. monitor the detection of the doorway-frame passing each of the side-mounted proximity sensors using *FramePassing*;
4. resolve any mis-alignment of the robot to the door frame using *Shift*;
5. monitor the movement of the robot, using *MoveDetect* to differentiate between maneuvering pauses and low velocity stagnation. Often during low velocities, the wheel motor drives will freeze up and require a reset of the velocity register to resolve.

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 triggers *MoveThrough* to also declare the successful achievement of its goal.

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 yields a different sequence of actions. Examples which have caused variations during the experimental runs are: doorway openings of unexpected size;

door placements within the opening that make detection and passage difficult; different directions in which the door opens; doors in a corner of the room as opposed to the middle of a wall; ghost doors caused by sensor noise; unanticipated obstacles in the path of the robot.

Knowing that these problems can occur is encoded as domain knowledge within the LSAs. The robot uses this knowledge to decide what to monitor and how to respond with the best strategy. Strategies are designed to be independent of the environment geometry whenever possible. Having the door, furniture, and robot located in different locations has little effect on the accomplishment of the goal, though it may have an effect on the selection of the strategies to apply.

5 The Experiments

The laboratory experiments consisted of three parts: (1) development of the implementation on a real robot; (2) knowledge acquisition and refinement through *guided 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, knowledge would be introduced to monitor for the occurrence of misalignment and to correct for it.

Test Set	Door Type	Door Size	Obstacles	Comments	#Goal	#Fail	%Goal
1	straight	100 cm	none	slightly wider doorway	4	1	80%
2	straight	100 cm	none	slightly wider doorway	4	1	80%
3	straight	81 cm	none	average size doorway	5	0	100%
4	straight	81 cm	none	average size doorway	4	1	80%
5	straight	70 cm	none	door too small	4	1	80%
6	straight	60 cm	none	door too small	5	0	100%
7	wall-out-right	81 cm	none	actual doorway	3	2	60%
8	wall-out-right	81 cm	none	actual doorway	4	1	80%
9	wall-out-right	81 cm	2	actual doorway	3	2	60%
10	wall-out-right	81 cm	2	actual doorway	3	2	60%
Totals	2 Door Types	4 Sizes	none and 2	50 Total Runs	39	11	78%

Figure 2: Results of capabilities experiments. We performed five runs for each test set.

However, we did not use in the guided experiments all the conditions we expected to find in the capabilities experiments. This allowed us to validate the robustness of the approach

when faced with situations not previously encountered. None of the guided experiments were conducted with doors too small, nor with the door type "wall-out-right", where walls are to the immediate right of the doorway. Some obstacles did come into play (rarely) and door sizes ranged from 81 cm to 95 cm.

The primary difference between the guided experiments and the capabilities experiments is that the knowledge engineer was allowed to interact with the progression of the experiment. This allowed specific interactions between the environment and the robot to be isolated, and thus allowed better incremental enhancement of the implementation (enhancement was *not* allowed during the capabilities experiments, and all capability experiments were allowed to run until failure or success).

Once a desired level of knowledge had been placed into the system, the system was tested with the capabilities experiments. A total of 50 capabilities experiments were run with 10 different scenarios of five runs each (see Figure 2). The scenarios varied in distance traveled to goal, doorway size, doorway type, obstacles, and status of the door. Test sets 1 through 6 were with a doorway constructed of cardboard boxes, which allowed to change its position and size[2]. Sets 7 through 10 were with a real door that opens out and to the right.

Of the 50 capabilities experimental runs, 78% completed successfully. 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. This 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. The clearance between the robot and the door frame in Test Sets 7 through 10 is very limited, as shown in Figure 3, and requires precision in sensing and control beyond any of the individual sensors available on our robot. The experiments provide evidence of the robustness of the approach.

Of the eleven cases that did not succeed, 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 crossing the doorway but not recognizing it, A detailed analysis of the experiments is in [Budenske, 1993].

The robot's performance in dealing with ghost doorways was extremely good. The door detection LSA often detected ghost doorways in the middle of the room or on walls where no doorway existed. In those cases, the robot would approach the ghost doorway and start its verification procedures. In all the experiments the robot determined that the doorway did not exist and resumed its search for a doorway.

One unexpected observation was the problems the robot had maneuvering near the edges of the doorway. If the robot ever ended up facing directly towards one frame of the doorway and with the other frame against its side between its front and rear bumpers, it would be in serious trouble. Most of the user interventions resulted from this situation. The development of specific domain knowledge for the detection and resolution of that situation could resolve it.

[2]Boxes were used to avoid potential damage to the robot due to collisions.

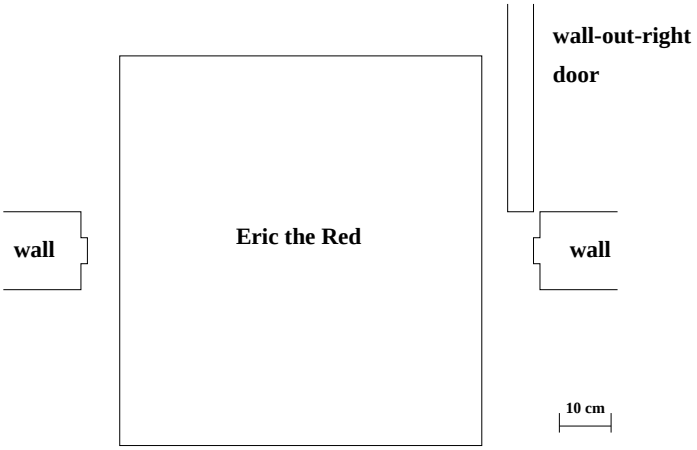

Figure 3: The clearance when traversing the door used in test sets 7 through 10 is limited. The figure is to scale.

Another unexpected observation was the difficulty the robot had 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 to the opening. At that close range, DetectDoor would loose track 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 eventually detect the doorway and start the process again. It would often take a number of repeated tries until the verify phase occurred far enough away from the opening to allow the detector to complete its job (persistence does pay off). The knowledge encoded within the LSA performed well at re-attempting to verify what appeared to be the correct door. This is a good example of how proper application of knowledge improves overall performance.

A systematic analysis of the properties of ultrasonic sensors, such as that reported in [Cox and Leonard, 1994], would produce additional knowledge that, when incorporated in the appropriate LSAs, would likely improve performance at achieving the navigation goals. Our interest has mainly focused on the overal design of the architecture more than on the specifics of the task.

6 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 [Brooks, 1986] 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 [Arkin, 1989]. 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, so providing for more flexibility.

The 3T architecture [Bonasso *et al.*, 1992] is an evolution of the architecture proposed by Gat [Gat, 1992]. The architecture 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 [Firby, 1987], while the lower level is a stateless reactive control mechanism which controls activities with no decision-making. Both this research and our work aim at issues of combined reactive and goal-directed behavior, sensor noise, actuator errors, and robust performance.

The primary difference is that our architecture is very homogeneous. Everything is an object and common methods are shared through inheritance. We have utilized the concept of Logical Sensor, proposed originally by [Henderson and Shilcrat, 1984], as the basis for constructing objects, and have extended it to include Logical Actuators and more sophisticated processing and reasoning.

Firby's most recent work [Firby, 1994] extends his original 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, "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 [Simmons, 1994] is another successful example of combining deliberative and reactive control. A large number of experimental results obtained using TCA with various robots and tasks attests the validity of developing flexible and reconfigurable systems.

Albus and his group [Albus, 1990] have developed a hierarchical, highly-synchronized control structure for the sensors which processes the sensor data as they flow though the structure. The highest level decisions are carried out in the top module, where the longest

planning horizons exist. The system, thus, is a large, highly-synchronized, static configuration of routines and subroutines which is not only imposed on the inter-module structure, but on the control and data paths, and the overall goal decomposition strategy. Our architecture allows dynamic instantiation of objects and reconfiguration of data and control paths among them.

There are similarities between LSA and blackboard architectures [Hayes-Roth, 1995]. 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 each utilizing the Sensor Explication algorithms. The storage of data for the blackboard architecture is centralized on the blackboard. The LSA requires data produced by a LSA to be stored within that LSA.

7 Conclusions

The Logical Sensor/Actuator architecture was developed as an approach to robotic execution of classical planning goals. This architecture addresses the issues of using up-to-date information to deal with a changing and partially unknown environment, the need to identify what information is relevant to achieve the goal, the need to handle noisy sensors and actuators, and, most important, the application of knowledge to resolve failures. Any interaction within the plan execution system is viewed as one of identifying and obtaining relevant information, including feedback information from processes involved in actuation.

The LSA theory contends that an abstract goal can be achieved through proper application of knowledge. Specifically, the procedural knowledge of how to "make explicit" the details abstracted during planning, and configure them into a process which achieves the abstract goal. These details are: (1) what information is relevant to achieving the goal; (2) what are the sources of this information. Extensive lab experiments demonstrate the viability of our method.

Acknowledgments

We would like to thank Scott Brandt, John Fischer, Chris Smith, Karen Sutherland, and Eric the Red for their help and efforts. This work was funded in part by the NSF under grant NSF/CDA-9022509, and by the AT&T Foundation. Though John Budenske is currently an employee of Architecture Technology Corporation, all research described in this paper was performed at the Artificial Intelligence, Robotics and Vision Laboratory of the Department of Computer Science, University of Minnesota.

References

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

[Arkin, 1989] R. C. Arkin. Motor schema-based robot navigation. *International Journal of Robotics Research*, 8(4):92–112, August 1989.

[Bailin, 1989] S.C. Bailin. An object-oriented requirements specification method. *Comm. of the ACM*, 32(5):608–623, May 1989.

[Bonasso *et al.*, 1992] 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.

[Brooks, 1986] Rodney Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1):14–23, March 1986.

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

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

[Cox and Leonard, 1994] I. J. Cox and J. J. Leonard. Modeling a dynamic environment using a multiple hypothesis approach. *Artificial Intelligence*, 66(2):311–344, 1994.

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

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

[Gat, 1992] 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*, pages 809–815, 1992.

[Hayes-Roth, 1995] Barbara Hayes-Roth. An architecture for adaptive intelligent systems. *Artificial Intelligence*, 72(1-2):329–365, 1995.

[Henderson and Shilcrat, 1984] T. Henderson and E. Shilcrat. Logical sensor systems. *Journal of Robotics*, 1(2):169–193, 1984.

[Simmons, 1994] R. Simmons. Structured control for autonomous robots. *IEEE Transactions on Robotics and Automation*, 10(1):34–43, February 1994.