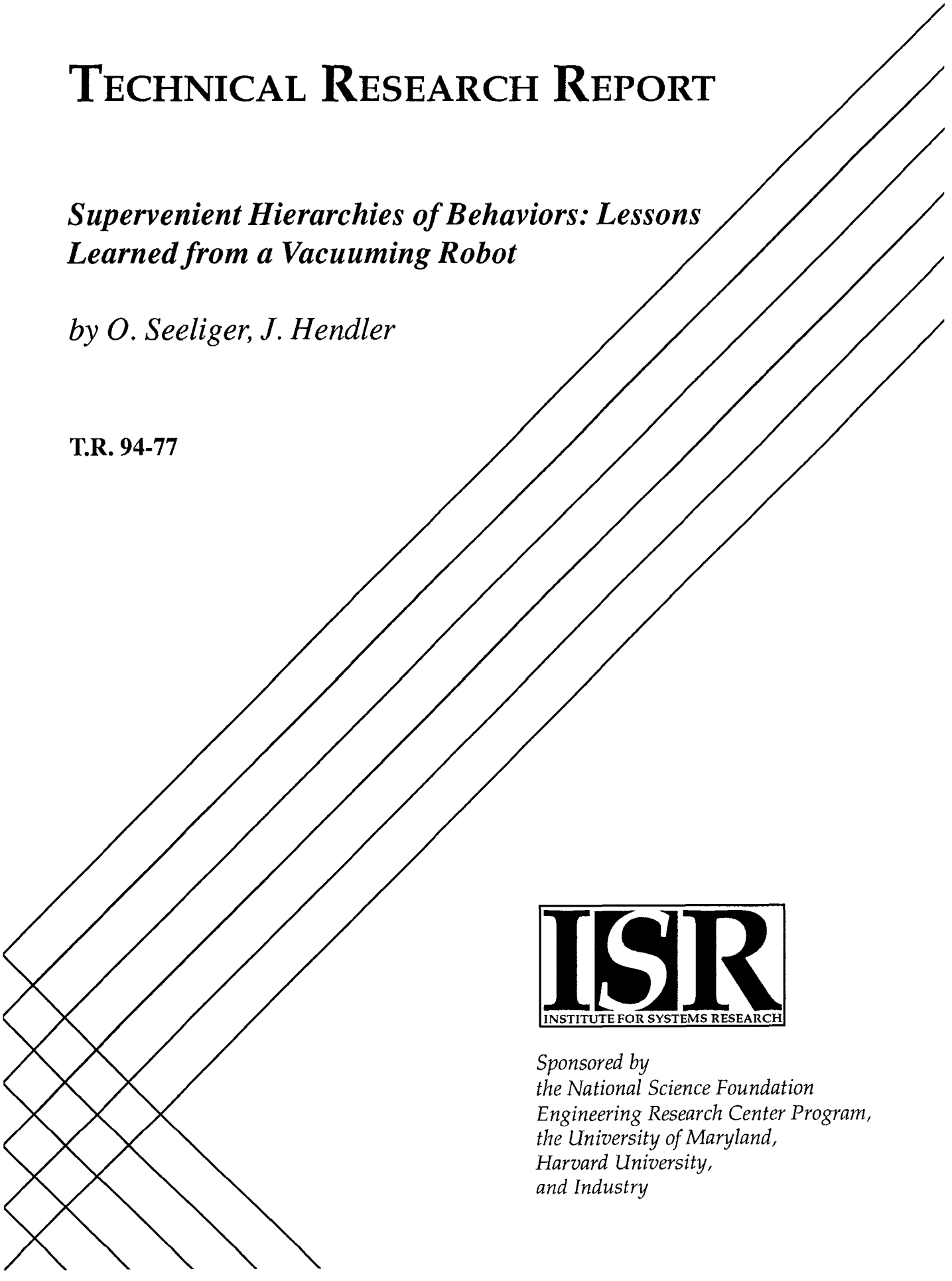
TECHNICAL RESEARCH REPORT

Supervenient Hierarchies of Behaviors: Lessons Learned from a Vacuuming Robot

by O. Seeliger, J. Hendler

T.R. 94-77



Sponsored by
the National Science Foundation
Engineering Research Center Program,
the University of Maryland,
Harvard University,
and Industry

Supervenient Hierarchies of Behaviors: Lessons learned from a Vacuuming Robot *

Oliver Seeliger James Hendler
Department of Computer Science
University of Maryland
College Park, MD 20742

October 27, 1994

Abstract

In this paper we describe the use of *behavior hierarchies* based on "merging" two models of multi-layer architecture — the supervenience model of [1] and the subsumption model of [4]. The behavior hierarchy approach allows us to use the robustness of reactivity in behavior design. It also encourages the design of modular behaviors that can be reused or more importantly recalibrated in different situations. We argue that behavior hierarchies extend our ability to design and program effective solutions that do not require any explicit planning. We also describe ongoing work in using this model for integrating planning and reacting This work is being used in support of an implemented system in which an autonomous mobile robot performs a vacuuming task.

1 Introduction

In Previous work [1], we described the *supervenience architecture,* a multilevel approach to integrating planning and reacting. We are currently exploring the use of this architecture to control an autonomous robot vacuum-cleaning system and to illustrate the use of supervenience as an architectural principle for building robust and intelligent behaviors.

The supervenience architecture is organized around a paradigm in which multiple levels pass information during problem solving. Higher levels model the world more abstractly, and are not permitted to sense the world directly. Instead, information about the robot's behaviors propagate up from the lowest level, while the higher levels initiate, monitor, and modify lower-levels based on the system's goals. This approach is summed up in the phrase "world-knowledge up/goal-knowledge down" which guides planning and reacting in the supervenience model.[1]

*This research was supported in part by grants from NSF(IRI-9306580), ONR (N00014-J-91-1451), AFOSR (F49620-93-1-0065), the ARPA/Rome Laboratory Planning Initiative (F30602-93-C-0039), and the ARPA I3 Initiative (N00014-94-10907)and by ARI (MDA-903-92-R-0035, subcontract through Microelectronics and Design, Inc.) Dr. Hendler is also affiliated with the UM Institute for Systems Research (NSF Grant NSFD CDR-88003012).

[1]This is, of course, an extremely short summary of superveniece. For details and for a comprehensive survey of how supervenience compares with other multilevel robotic approaches see [1, 3].

The original work in the development of the supervenience architecture was performed using a simulator which made many simplifying, and unrealistic, assumptions about the robotic behaviors which were available. To make the model more realistic, a small robot was purchased, and an attempt is being made to use this architecture in designing the robot's behaviors and the planning system they interact with. So far, our main work has involved developing reactive behaviors on the robot with little interaction with the planner. Therefore the emphasis of this paper is on the robot's part of the system.

More specifically, we introduce the concept of *behavior hierarchies* which combines ideas from Brooks' subsumption architecture[4] with our supervenience architecture model. Since the low-level behaviors of the robot are close to the world, and the need for abstract symbolic representations is small, we do not need the full capabilities of the supervenience architecture. On the other hand, a purely subsumption-based system, where so-called high-level behaviors are only allowed to gate (i.e. turn on and off) low-level behaviors, is more restrictive than allowed in using supervenience. We therefore have been using a model that is partway in between. It assumes purely subsumptive behaviors at the lowest levels, communicating with higher level behaviors that can not only gate the lower-levels, but can also change parameters, modes, and otherwise modify the lower level behaviors so as to better achieve the more abstract goals these levels attempt to maintain.

In section 2, we give a detailed description of the vacuum-cleaning task being performed by the behavioral hierarchies on our current robot. Section 3, outlines the system used to solve the problem and defines the principle of behavior hierarchies. Section 4 then demonstrates the particular behavior hierarchies for the vacuuming task in detail, and outlines the role of each individual behavior in the hierarchy. We conclude with a brief description of how the planner will be added, to allow a higher level of problem-solving built on the behavior hierarchy base.

2 Vacuuming

Two factors dominate vacuum cleaning in real life. One of them is uncertainty in the world. A typical room usually has a variety of objects with varying shapes, orientations, and locations. A table that was in the center of the room yesterday might have been moved to a wall today. Also, room shape differs throughout a building, doors can open and close further altering the room's base shape, etc. This produces an uncertain environment, in which a vacuum-cleaning system must be flexible enough to deal with unexpected events.

The other challenge in the vacuum-cleaning arena is other "dynamic" entities, such as humans or pets. A vacuum-cleaning robot has to avoid hitting dynamic entities much like static objects. However, this constraint is a lot more time-critical. Moreover, a system needs to distinguish dynamic obstacles from static ones in order to vacuum-clean satisfactorily. For example, if a robot encounters a human, it might have to decide to vacuum another area first before coming back, hoping the human has moved from that location in the meantime, so that the robot can resume from there.

For our system, we define the vacuum-cleaning arena as a rectangular area (although the vacuum-cleaning process can also be used for more general areas). This area can contain static objects (obstacles) and dynamic entities. The goal is to sweep the entire area that is not blocked by static obstacles. The process is straightforward when there are no obstacles

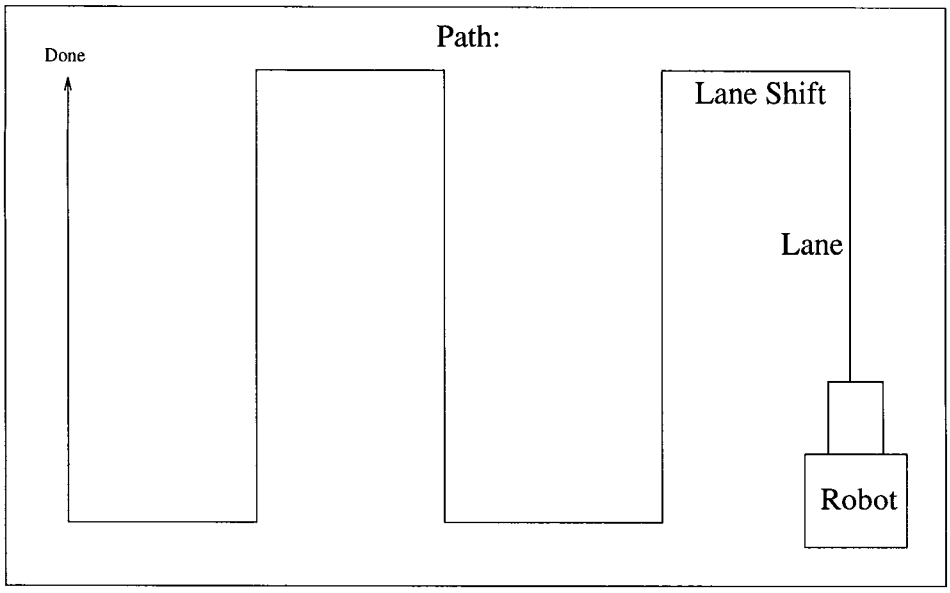


Figure 1: Rectangular Vacuum-Cleaning Arena

in the arena. Figure 1 shows the path of a robot in such a scenario. Conceptually, we divide the arena into lanes. After positioning itself in a corner, the robot moves along these lanes. The robot turns around and switches lane whenever it reaches the end of the arena. Using this process, the robot sweeps the entire area.

In general, the robot always attempts to use this process to solve the task. Whenever something unexpected happens, the robot tries to improvise. Figure 2 shows how the presence of an obstacle causes the robot to modify its path structure on the fly. The system has to temporarily overrule the main goal to sweep the entire area with the goal to move around the obstacle. Therefore the robot follows a new path until it first circumnavigates the obstacle and secondly repositions itself in the original lane. Then the main process can resume. The shape of obstacles can be arbitrarily complex, with the complexity of the improvisation effort increasing with the complexity of the obstacles' shape. Compare the scenario in figure 2 to figure 3.

In spite of the complexity involved, a robot can master many vaccuming situations without the aid of a planning system. Consider the domain consisting of the set of all static obstacles of rectilinear base shape, i.e. a polygon only having right angles such as the obstacle in figure 3. There exists an effective way to move around such obstacles. When the robot encounters an obstacle we arbitrarily pick a direction in which to circumnavigate it, say to the right. After the initial 90-degree right turn, the robot moves forward tracking the obstacle's "front" side. From then on the algorithm is simple. Whenever its left side becomes free (meaning the robot reached the end of the obstacle's side), the robot turns left 90 degrees and resumes moving forward. Whenever something blocks the robot's front side, the robot stops and makes a right 90-degree turn and again resumes moving forward (This situation occurs when the obstacle has an arm-like extension). This process terminates with

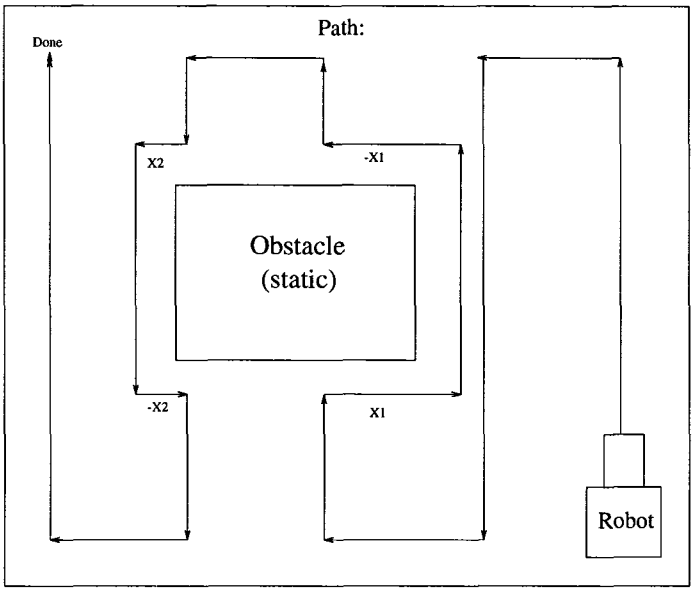


Figure 2: Vacuum-Cleaning Arena with Obstacle

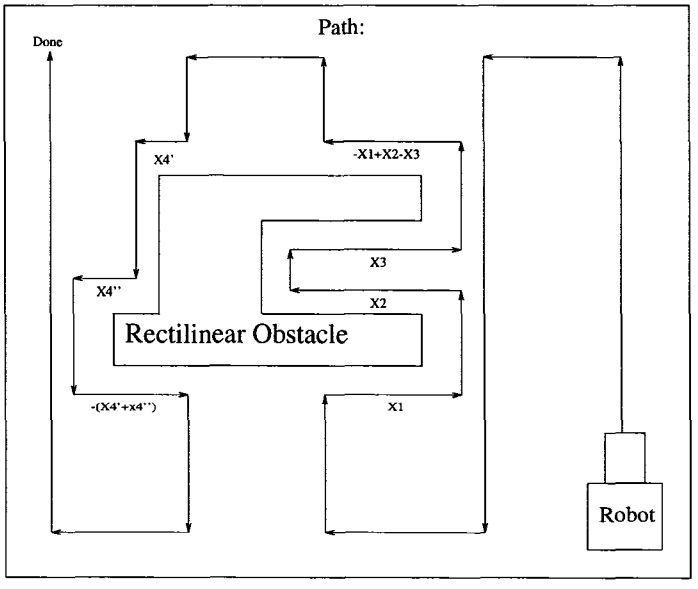


Figure 3: Vacuum-Cleaning Arena with Rectilinear Obstacle

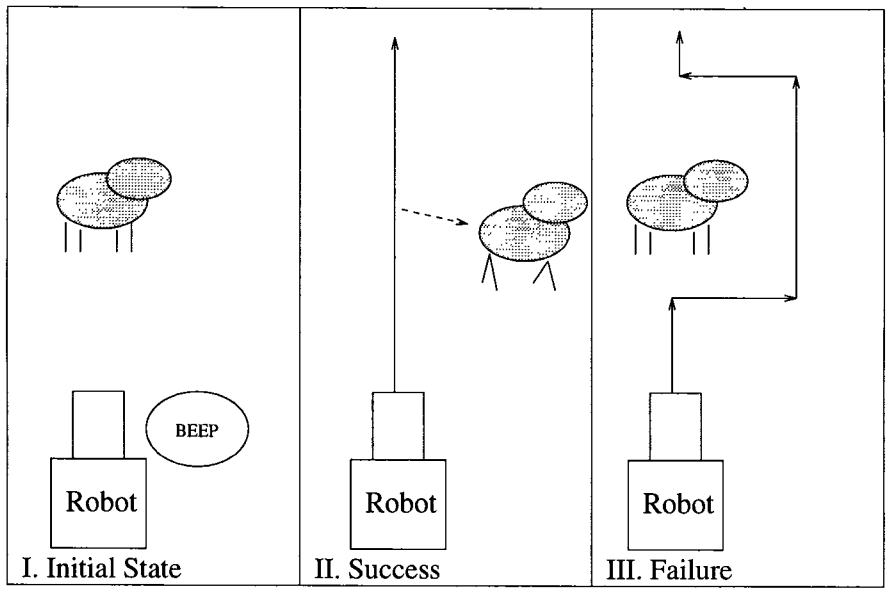


Figure 4: Approach to Chase Away Dynamic Entities

a final right turn as soon as the robot reaches the original lane.

This shows that there are successful solutions addressing the uncertainty in the world involved. Note that although a planner isn't necessary, it would be advantageous to use one. For example the decision to circumnavigate the obstacle to the right could be made by a planner based on a map. The planner could tell the robot, for example, which way is likely to be shorter or if one direction leads to an occluded path, etc. This decision is based on goals that the planner has (to move around the obstacle as efficiently as possible or to cover as much of the floor as can be reached) yet filters down to the robot, through the supervenient levels, to become a concrete activity (for example, a left turn).

The other factors we need to address are dynamic entities in the arena. While again a planner would be useful for reasoning if the robot's path is blocked by something static or dynamic, we currently solve this problem without a planner. Based on the premise that a person or a pet would move when given a signal, our robot can produce a shrill sound to chase away whatever dynamic entity might block its way. A simple solution is to wait for a while. If nothing moves, the system consider the obstacle unmovable, i.e. either static or dynamic, but not willing or likely to move out of the way. As illustrated in figure 4, in the latter case, the robot treats the object like a static one and attempts to move around it.

2.1 Rectilinear Obstacle Avoidance

As we've mentioned, we have severely limited the world by only considering avoidance of rectilinear obstacles. In fact, it turns out that such a restriction is both important to our ability to solve this problem with a low-cost, imprecise robot and also less of a restriction in practice than one might think. As these features turned out to be an important aspect of

the problem which we've learned about specifically through implementation of the system on a real robot, we will digress for a moment and explore this issue.

The original reason for basing avoidance on rectilinear objects was that the robot's capabilities are very limited. Its sensing are limited to infrared and bump sensors and the motor control does not produce turns that are particularly close to 90-degree turns. Furthermore, the robot doesn't have a global positioning device. This limits the reliability of the robot's performance and adds further uncertainty. Since a "turn 90 degrees" behavior rarely results in an actual 90-degree turn, the accumulated error of repeated 90-degree turns represents a severe interference with the robot's goal to sweep the entire arena.

To some extent, our behavioral repetoire was shaped by our need to deal with this uncertainty in the robot's performance. For example after the robot completes a lane in the rectangular vacuum-cleaning arena, it backs up toward the arena border in order to realign itself. This process stops as soon as the robot senses the arena wall on two different points on its back, which ensures that the robot's direction is oriented perpendicularly to the wall. In practice, we have observed corrections of as much as +/-30-degree angles. That is, without realigning itself with respect to the arena's border first, the robot would in extreme cases head in a direction 30 degrees off from the modeled direction. Thus, due to the difficulty in making turns we felt that we had to restrict the objects to simple rectangles.

We later realizd that what worked for rectangles would work relatively well for arbitrary rectilinear obstacles *if* we could add behavioral control which allowed us to cope with the deviances between expected states (e.g. the angle with respect to the border) and actual states. To a large extent, the reactive behavior hierarchy, which we describe later in this paper, grew out of our need to compensate for such discrepancies. For example, a reactive wall-following behavior tracking the side of a rectilinear obstacle needs to continuously maintain the same distance to the wall. Therefore, we add a hierarchical behavior which causes the robot to periodically orient itself with respect to the obstacle, such that the direction of its moves is constantly corrected to be closer to the modeled direction.

It turns out that there is a second argument in favor of defining the world in terms of rectilinear obstacles. In practice, the restriction turns out to be reasonable to place on the robot's behavioral system, even where obstacles in the world are not rectilinear. This is because nonrectilinear objects, or rectalinear objects in non-90-degree orientation with the robot, can be approximated by the robot using rectilinear motions (see figure 5). The accuracy of this approximation actually depends more on the features of the robot than on the featurse of the obstacle. Thus, each line segment of the rectilinear approximation will be based on how closely the robot can sense it. For example a circular base shape (cylinder) can be modeled as a square or a cross depending on the size of the cylinder relative to the sensing capabilities of the robot.

Also, an obstacle in the real-world might not be aligned parallel to the vacuuming arena or to the direction in which the robot is vacuuming. Using our approximation, we can also model an object that is rectilinear, but oriented differently with another rectilinear base shape that is oriented with respect to the arena (figure 6). Therefore, although formally the domain is restricted, the robot can deal with a fair amount of objects not specifically allowed in the domain model by handling the objects as approximations to obstacles that are in the domain.

Again, however, for such an approach to work, the robot must be able to modify its

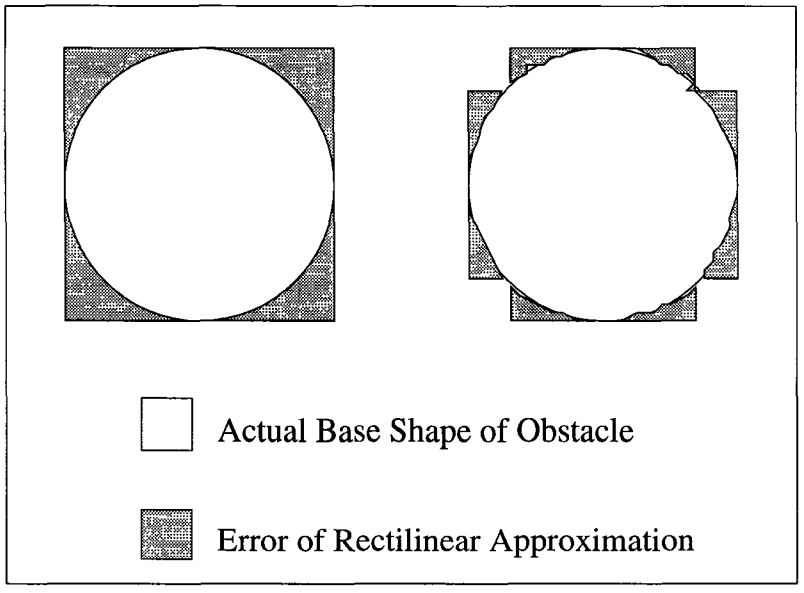


Figure 5: Two Rectilinear Base Shape Approximations

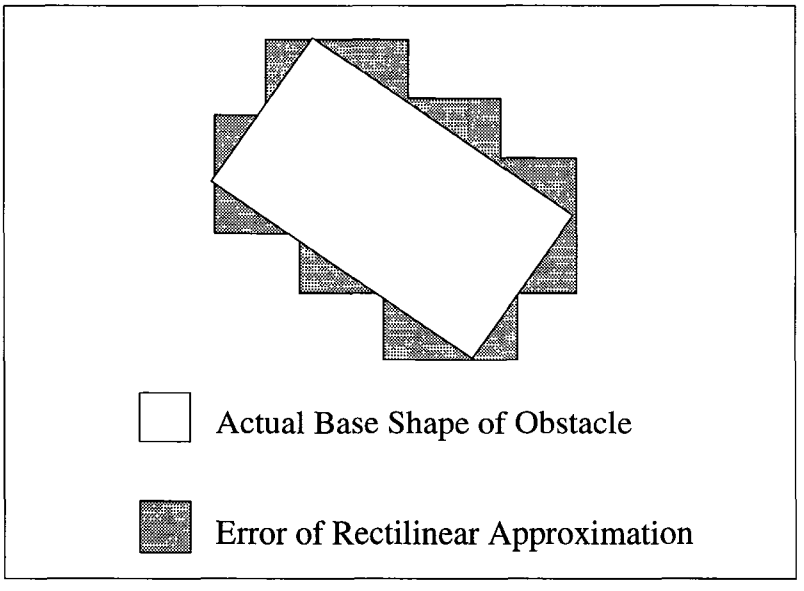


Figure 6: Oriented Rectilinear Approximation of Obstacle

behavior so that the approximation is what is actually realized. Thus in Figure 6, for example, the robot must make a number of 90-degree turns based on sensing values. A higher-level behavior that can be informed of divergence from an expected value (that the side sensor will be reporting a near constant value) and which can modify the underlying behavior accordingly (suggesting additional turns or tracking) makes the robot's motions meet such an approximation.[2]

Thus, the uncertainty of the robot's control and the need for a simple approximation that, with appropriate dynamic modification, can handle many more complex real-world situations made it clear that something beyond simple behavior-based control was needed. A hierarchical system, in which there existed behaviors that could make decisions as to how to gate and/or modfy lower level behaviors, seemed to be necessary. The communication between the behaviors in such a hierarchy were more complex than those provided by the subsumption model but simpler than those allowed in the full supervenience system (where symbolic descriptions may be propagated and reasoned about). As such, experiments with this domain led to the development of a sort of "hybrid" model, somewhere in between the two.[3] In the next section we discuss this approach, and the details of its use for the vacuuming robot.

3 System Architecture

As mentioned earlier, our system currently runs using two hardware components, a robot programmed in a subsumption architecture language which communicates with a planner running remotely. Our robot is an ISRobotics R2-e with eight infrared sensors as well as eight touch sensors, and two wheels driven by motors with shaft-encoders. It also has a 2-DOF gripper, in which we mount a small, protable vacuum cleaner. (The vacuum is switched on before running the system so that the robot cleans that area which it traverses while maneuvering across the arena.) The planner, still in its developing phase, runs on an Apple Macintosh and is based on Spector's supervenience architecture. To link the Macintosh to our R2e, a wireless modem communicates between the planner and the robot.

Our original goal was to implement the planner using the supervenience system, and to have it directly call behaviors running on the robot, since it was clear that the R2e's subsumption architecture could not support many of the features the supervenience architecture allows. However, as alluded to in the last section, we discovered a need for a hierarchy of behaviors to be used. Thus, we have been exploring the use of behavior hierarchies, which themselves borrow from the supervenience approach, but essentially work on a simpler scale. (We note that just as problem solving in subsumption may result from several separate, usually concurrent, behaviors, several behavior hierarchies may be used in complex problem solving. In our vacuuming system, only a single hierarchy is used.)

In the remainder of this section, we describe the behavior hierarchy approach in more detail, and show how it is used in the vacuuming robot.

[2]This is a clear example of what Brooks calls the world being its own best model[4].

[3]Thus, depending on one's world view, behavior hierarchies can be viewed simply as an extension to subsumption, a simplification of supervenience, or a separate "hybrid" model. In the remainder of this paper, we treat them as the latter, but we do not feel it is a particularly important distinction.

3.1 Behavior Hierarchies

The central design feature in our implementation of behavior hierarchies is borrowed from that of supervenience – the separation of levels based on "world-knowledge up/goal-knowledge down". Thus, we design communication such that the lower a behavior is in the hierarchy, the closer it should be to the world.[4] Naturally, the closer a behavior is to the world, the more concrete are the elements the behavior senses or manipulates. By contrast, the higher a behavior is in the hierarchy, the further away it is from the real-world and the more abstract are the behavior elements. In turn, "goal-knowledge down" requires that behaviors at a higher level are, essentially, closer to the goals of the system rather than anchored in particular sensory values. As goal-based information is passed to lower levels, it is decomposed and reformulated to result in a description closer to the world than the previous one. In this process goal-knowledge loses importance while world-information becomes more important.

Behavior hierarchies are extensions of subsumption architectures which assume a collection of behaviors not sharing any global variables. Instead, interaction between behaviors occurs through "wires" (typically implemented in software, rather than hardware). Each behavior has input and output wires, and output wires of behaviors are connected to input wires of other behaviors. All behaviors run concurrently sending back and forth information if necessary. Behaviors may also activate and deactivate other behaviors.

Behavior hierarchies work by imposing a partial order on the collection of behaviors. In this order, every behavior that precedes another behavior is, essentially, more abstract than the other behavior, i.e. more distant from the world (and, in fact, it cannot directly sense the world, but only receive inputs from the levels closer to the world). The communication between the levels allows higher-level behaviors (those preceding others in the order) to send messages which either activate, deactivate, or change parameters of the lower level behaviors. The lower-level behaviors send messages which report "events" to higher-levels. These events typically do *not* consist of specific sensor values, but rather values representing initiation, completion, or errors arising within the behavior. Thus, a particular value might represent that an obstacle was encountered while moving in a certain direction.

We should note here that the term level is somewhat misleading because no two behaviors are explicitly "on the same level." The only defining characteristic of a behavior hierarchy are the relationships explictely defined by the partial order. The communication protocols, which allow one behavior (higher level) to modify another (lower) level, but which only allow the lower to 'inform" the higher, realize the asymmetric leveling which characterizes supervenience.

Since the behavior hierarchy's place is to be the lower part in the global supervenience architecture, all behaviors are relatively close to the world. This means that typically the level of the abstractions and therefore the need for symbolic representations of the world is limited. Symbolic abstractions are not supported nor encouraged in Brooks' subsumption architecture. To allow the behavior hierarchies to take advantage of the strengths of reactive behaviors, we also try to refrain from using explicit symbols wherever possible. Thus, a chair might appear as sensor value 3 to a very low level, while it may be "object to the left"

[4] Note that while this is typical of subsumption architectures, it is not mandated in the design, and is often violated in complex subsumption-based systems.

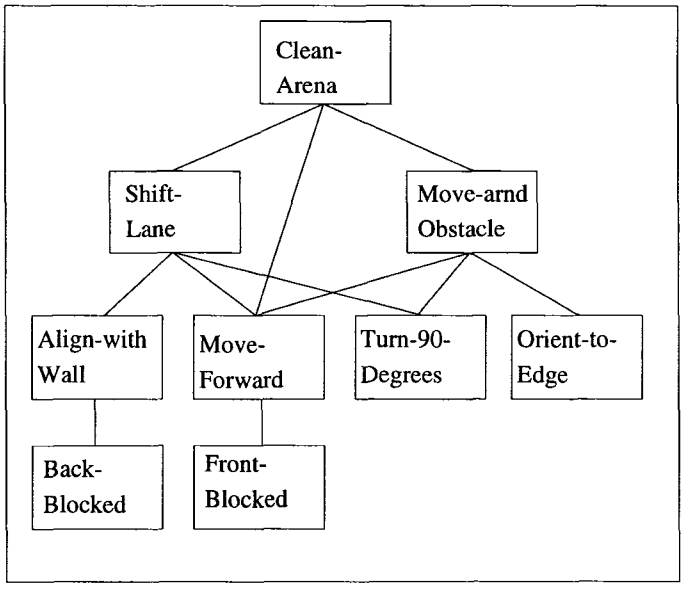


Figure 7: Behavior for Vacuuming with Rectilinear Obstacles

at a higher level. Thus, the transition from sensor signals to symbols occurs throughout the hierarchy, but the mapping to specific symbols like "chair" or "chair-1" will only occur in the planner.

It should also be noted that an explicit aspect of the supervenience approach is that a problem should be resolved at the lowest level possible. Information about the event may then propagate up and eventually change goals (and therefore cause information to propagate down and change behavior), but that doesn't imply that the system stops while this occurs. Behavior hierarchies also allow this, letting lower-level behaviors continue to run until a new signal (to change behavior or deactivate) is received from above.

3.2 The behavior hierarchy for the vacuuming robot

To illustrate behavior hierarchies in more detail, we return to the vacuuming robot. Figure 7 presents part of the behavior hierarchy we use to perform vacuuming without the aid of a planner. It should be clear that this behavior hierarchy is a partial order. For example, while the behavior pairs "Clean-Arena" and "Move-Forward", or "Shift-Lane" and"Back-Blocked" are ordered with respect to each other there is no such relationship between, for example, "Move-Forward" and "Turn-90-Degrees."

To see how this hierarchy works, consider how normal vacuuming proceeds. In this hierarchy, the only behavior without a higher behavior is "Clean-Arena" and it is invoked first to start vacuuming. As the name implies, the goal of this behavior is to vacuum the entire area. This behavior in turn causes the lower-level behavior "Move-Forward" to be activated. While "Move-Forward" is active, a command is issued to start the robot moving at a given speed and to keep track of the distance moved. It also activates the "Front-

Blocked" behavior. "Front-Blocked" polls the front infrared sensors to detect objects in front of the robot. After detecting an obstacle, "Front-Blocked" sends the message to "Move-Forward" that something is in front of the robot. "Move-Forward" immediately stops the robot to avoid a collision. Simulataneously, it also passes a message up to "Clean-Arena" that tells that it is blocked and states how far it moved prior to the blockage. Depending on the distance moved, "Clean-Arena" then decides whether the robot has reached the end of the platform or an obstacle. Based on that decision it will send the message to start either "Shift-Lane" (which turns the robot around) or "Move-around-Obstacle" (which navigates the robot around the obstacle using the methods described earlier).

The notion of abstraction, as supported in the supervenience architecture, is also realized in the behavior hierarchy, as can be seen by looking more closely at the vacuuming behaviors. Consider the relation between the lower-level "Move-Forward" and its higher-level "Clean-Arena." Forward is a direction directly realizable by the robot as a particular way to move the wheels. "Arena," on the other hand is not a directly sensable object. Instead, a preprogrammed set of parameters (these may in turn be changed by the planner when the implementation is extended), is used to correspond to a range of directions, distances, etc. These cannot be directly sensed, but can be "reasoned" about as a combination of other sensor values reported up by the lower levels. Similarly "Back-Blocked" is a lower level since "something behind the robot" is a real entity which directly effects a sensor, while "lane" is an abstraction used to represent a potentially large set of possible sensing and effecting actions.

To be precise, however, we wish to make it clear that while the behavior names contain symbols such as"arena","lane", and"obstacle" to prevent confusion, the behaviors themselves don't represent any such symbols internally. "Shift-Lane," for example, is itself a behavior, and thus it doesn't work by invoking rules or other such to directly reason about "lane"s. Instead, it has as its goal simply to invoke a short forward move between two 90-degree-turns followed by realigning the robot with whatever is behind it. Thus, the abstraction, while real and useful to the programmer, is not directly represented in the behavior hierarchies. Rather, it is enforced by the "supervenient" information flow in which only the lower levels have direct access to the sensors, and higher levels can only base their effects on the events reported up.

As an example of how world knowledge is filtered up in the form of events, consider what happens after "Clean-Arena" invokes the behaviors "Shift-Lane","Move-Forward", and' 'Move-around-Obstacle" (thus converting its goal into instructions for the lower levels). "Move-Forward" then invokes "Front-Blocked" and the robot moves until something is sensed closely in front of it. At this point, "Move-Forward" reports to "Clean-Arena" a message which, essentially, reports "I have moved forward x-amount and stopped because something is blocking the robot's path." At this point, "Clear-Arena" then reacts to this event, either invoking "shift-lane" (because the distance is far enough) or "Move-Around-Obstacle" (because it isn't).

To see the advantages of separating the high- and low-level behaviors to allow interaction between these different levels of abstraction, consider the "Move-around-Obstacle" behavior. First, let's consider how we designed this behavior simply to handle simple rectangular objects, as shown previously in Figure 2. The robot moves forward until blocked, and "Clear-Arena" invokes "Move-around-Obstacle." This behavior then initiates "Turn-

90-degrees" such that the robot turns to the right. After the robot makes the initial right turn, the obstacle will be on its left. As the robot moves forward, it reaches the end of a wall of the obstacle, and the left-side becomes clear. This event is reported back to "Move-around-Obstacle," which then orders the robot to move forward again until the obstacle is passed. At that point, another 90-degree turn (in the other direction) is initiated and so on until the robot has moved back to its original lane. The "Move-around-obstacle" behavior then reports successfully to its parent, and normal vacuuming continues.

We later decided to change this behavior to allow circumnavigation around more complex (arbitrary) rectilinear onjects, such as shown in in figure 3. To achieve this we were only required to make a simple change to "Move-Around-Object." In particular, we simply need to add an instruction that changes what happens when the front is blocked during the forward move back to the original lane. In particular, we simply needed to add an additional call to "turn-90-degrees" so as to instruct the robot to make an additional right turn. The remainder of the behavior is then the same, and thus it is as if the robot is simply going around a number of small regular rectangles.[5]

4 Adding a planner

As can be seen from the previous section, having the higher levels able to invoke, modify, or disable lower-level behaviors allows us to achieve a fairly complex overall behavior reactively – the robot does a reasonable job of vacuuming around relatively complex obstacles without the need to appeal to an explicit (path) planner. On the other hand, there are times when it would be extremely useful to have access to such a planner. For example, if we had a planner above "Move-around-Obstacle," it could make the decision as to which direction the robot should start to turn based on an explicit world map (with locations of known obstacles provided in advance and/or built out of the reports arising from the behaviors in this heirarchy) or on information about common obstacles. This would allow better decision making and more efficient vacuuming.

Conversely, if"Move-around-Obstacle" doesn't find an expected corner of an obstacle (i.e. the robot has encountered an unexpected wall), it could report this by sending an appropriate event report to the planner. The planner could then reason as to what had gone wrong and, perhaps, realize that "Clear-Arena" has erroneously decided that the robot has reaches an obstacle instead of the arena border. It could then revise the robot's strategy and order an alternate one. Given the inherent imprecision in even well engineered robots, and the incredible imprecision in the low-cost robot we use, this would enable significant improvement, particularly in more complex situations such as a maze-like office hall environment.

In addition, use of the planner could also be helpful in providing explicit guidance when the default behaviors are inappropriate. Thus, for example, the planner could monitor how well it is vacuuming around an obstacle by building a simple map based on the edge reports. If the planner decided that a rectilinear approximation was not appropriate, it could intervene and override the robot's"instinct" to apply this approximation, causing it

[5]This same change also enables the approximations to rectilinear for other obstacles, as described previously.

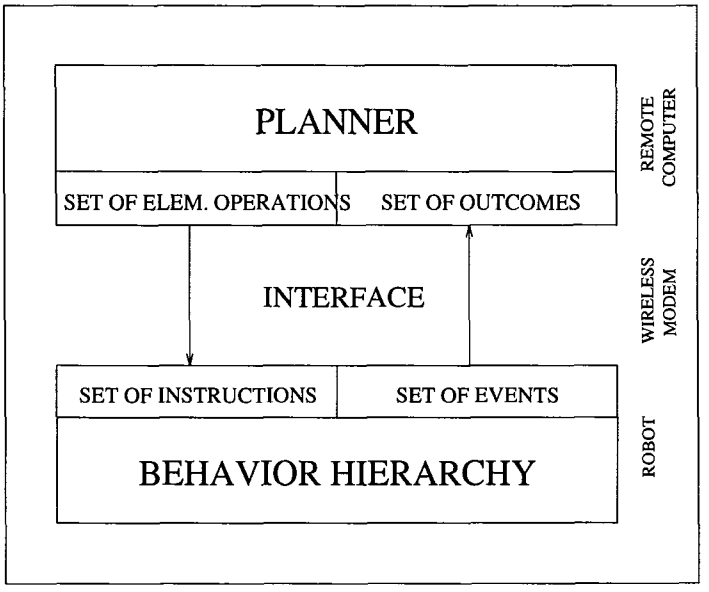


Figure 8: Overview of planner/robot communication.

to invoke a different strategy. For example for a robot with a better positioning system able to drive along arcs more precisely, the planner could decide to invoke a "circle-around" behavior if it realized a cylinder was blocking the path.

We are currently working on adding exactly such a planner to the system, as shown in Figure 8. The events reported by the highest level of the behavior hierarchy are broadcast to the planner running on a remote system.[6] The planner would have access to a set of elementary (a/k/a primitive) operators, which would correspond to the highest level behavior(s) in the behavior hierarchy. Using a dynamic planning model[7], the events reported by the highest levels of the behavior hierarchy correspond to sensing events that are at a high enough level of abstraction for the planner to use. Using such a model will, we hope, help to bridge the traditional gap between sensors and planners, and thus give us a strong base on which to experiment with the interaction between reactive and plan-based behaviors.

5 Conclusion

In this paper we have described the use of behavior hierarchies based on "merging" two models of multi-layer architecture — the supervenience model of [1] and the subsumption model of [4]. The behavior hierarchy approach allows us to use the robustness of reactivity in behavior design. It also encourages the design of modular behaviors that can be reused or more importantly recalibrated in different situations. In addition, we have argued that while behavior hierarchies extend our ability to design and program effective solutions that do

[6] In a larger robot, the planner would be onboard preferably running on a separate processor.

[7] We are currently working on a planner based on the DR model we describe in [5, 6].

not require any explicit planning, they also provide a useful model for integrating planning and reacting. We are currently exploring both of these directions. In particular, we are working on extending the behavior hierarchies for control of more complex robots solving harder problems than those seen in the simple vacuuming domain[8] and also extending the vacuuming robot by adding a planner. We believe this latter will let us realize the main goal of our earlier supervenience work – achieving an integration between reactive and deliberative problem solving strategies.

References

[1] Spector, L. and Hendler, J. "Planning and Control across Supervenient Levels of Representation," *I.J. Intelligent and Cooperative Information Systems*, 1(3-4), December, 1992 (w/L. Spector)

[2] Brooks, R. Behavior Language Manual, MIT Technical Report, 1992.

[3] Spector, L. "Supervenience in Dynamic World Planning" Doctoral Dissertation, University of Maryland, 1992.

[4] Brooks, R. Elephants don't play chess, *AI Journal*, 1990.

[5] J. Sanborn and J. Hendler "A model of reaction for planning in dynamic domains" *International Journal of AI and Engineering,* Volume 3(2), April, 1988.

[6] R. Kohout, D. Musliner, and J. Hendler, "Dynamic Reaction on the Maruti Hard Real-time Operating System, Technical Report, UMCP (in press).

[8] Particularly including some nascent work in adding a behavioral repetoire to a robot designed for space teleoperation work.