Chapter 3
Robot Behavior

The great end of life is not knowledge, but action.
—Thomas Henry Huxley

We really only know, when we don't know; with knowledge, doubt increases.
—Johann Wolfgang von Goethe

For in much wisdom is much grief, and he that increaseth knowledge increaseth sorrow.
—Ecclesiastes 1:18

Chapter Objectives

1. To learn what robotic behaviors are.
2. To understand the methods that can be used to express and encode these behaviors.
3. To learn methods for composing and coordinating multiple behaviors.
4. To obtain a basic understanding of the design choices related to behavior-based robotic systems.

3.1 WHAT ARE ROBOTIC BEHAVIORS?

After developing an understanding of behavior's biological basis in chapter 2, we now study how to express the concepts and formalisms of behavior-based robotic systems. It is important to remember that biological studies are not necessarily viewed as constraining for robots, but nonetheless serve as inspirations for design.

Perhaps the easiest way to view simple robotic behaviors is by adopting the concept advocated by the behaviorist school of psychology. A behavior, simply put, is a reaction to a stimulus. This pragmatic view enables us to express how a robot should interact with its environment. By so doing, we are confining ourselves in this chapter to the study of *purely* reactive robotic systems.

3.1.1 Reactive Systems

If we had more time for discussion we would probably have made a great many more mistakes.

—Leon Trotsky

> A **reactive** robotic system tightly couples perception to action without the use of intervening abstract representations or time history.

Reactive robotic systems have the following characteristics:

- *Behaviors serve as the basic building blocks for robotic actions.* A behavior in these systems typically consists of a simple sensorimotor pair, with the sensory activity providing the necessary information to satisfy the applicability of a particular low-level motor reflex response.
- *Use of explicit abstract representational knowledge is avoided in the generation of a response.* Purely reactive systems react directly to the world as it is sensed, avoiding the need for intervening abstract representational knowledge. In essence, what you see is what you get. This is of particular value in highly dynamic and hazardous worlds, where unpredictability and potential hostility are inherent. Constructing abstract world models is a time-consuming and error-prone process and thus reduces the potential correctness of a robot's action in all but the most predictable worlds.
- *Animal models of behavior often serve as a basis for these systems.* We have seen in chapter 2 that biology has provided an existence proof that many of the tasks we would like our robots to undertake are indeed doable. Additionally, the biological sciences, such as neuroscience, ethology, and psychology, have elucidated various mechanisms and models that may be useful in operationalizing our robots.
- *These systems are inherently modular from a software design perspective.* This enables a reactive robotic system designer to expand his robot's compe-

tency by adding new behaviors without redesigning or discarding the old. This accretion of capabilities over time and resultant reusability is very useful for constructing increasingly more complex robotic systems.

Purely reactive systems are at one extreme of the robotic systems spectrum (section 1.3). In subsequent chapters, we will see that it may be useful to add additional capabilities to reactive systems, but for now we focus on these simpler systems.

3.1.2 A Navigational Example

Let us construct an example with which we can frame the discussion to come. Consider a student going from one classroom to another. A seemingly simple task, at least for a human. Let's examine it more closely and see the kinds of things that are actually involved. These include

1. getting to your destination from your current location
2. not bumping into anything along the way
3. skillfully negotiating your way around other students who may have the same or different intentions
4. observing cultural idiosyncrasies (e.g., deferring to someone of higher priority if in conflict with priority determined by age or gender, in the United States passing on the right, etc.)
5. coping with change and doing whatever else is necessary

So what sounds simple (getting from point A to point B) can actually be quite complex (figure 3.1), especially in a situation where the environment is not controllable or well predicted.

Behavior-based robotics grew out of the recognition that planning, no matter how well intentioned, is often a waste of time. Paraphrasing Burns: The best laid plans of mice and men oft go astray. *Oft* is the keyword here. Behavior-based robotic systems provide a means for a robot to navigate in an uncertain and unpredictable world without planning, by endowing the robot with behaviors that deal with specific goals independently and coordinating them in a purposeful way.

3.1.3 Basis for Robotic Behavior

Where do robotic behaviors come from? This primary question leads to a series of subsidiary questions that must be answered to provide a robot with behavioral control:

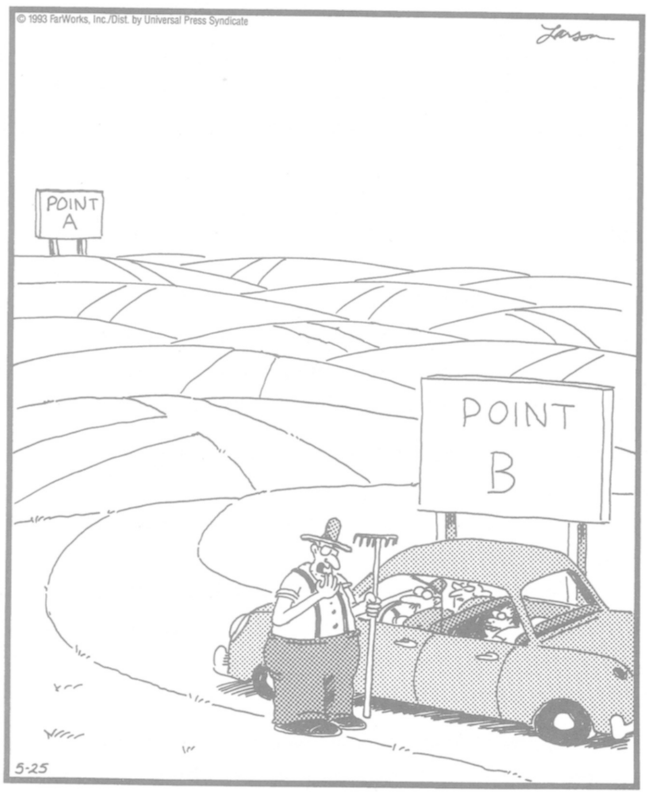


Figure 3.1
Things may be harder than they seem. (The Far Side © 1993 Farworks, Inc. Distributed by Universal Press Syndicate. Reprinted with permission. All rights reserved.)

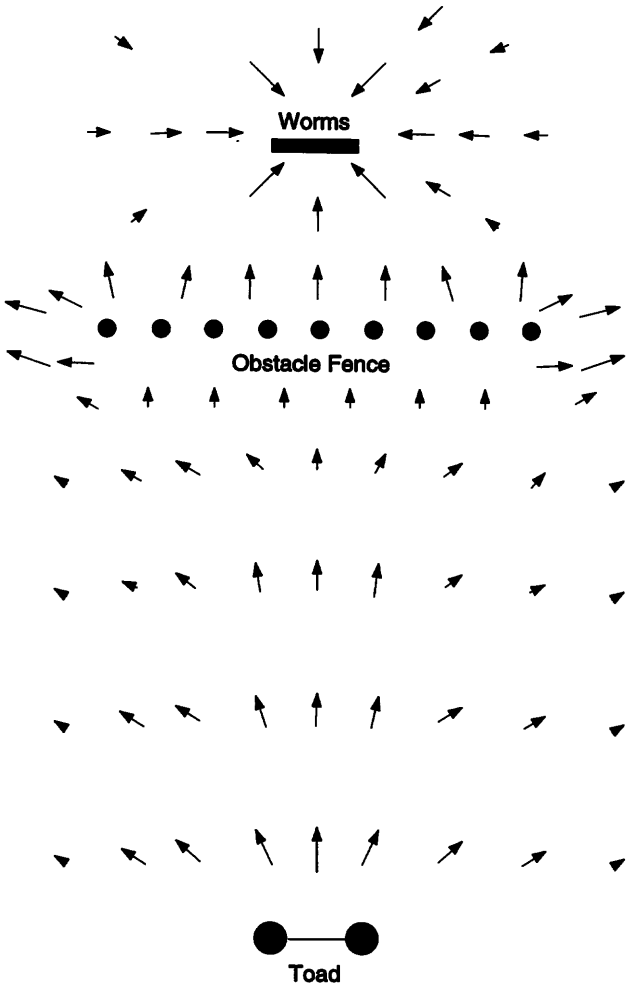
- What are the right behavioral building blocks for robotic systems?
- What really is a primitive behavior?
- How are these behaviors effectively coordinated?
- How are these behaviors grounded to sensors and actuators?

Unfortunately there are currently no universally agreed-upon answers to these questions. A variety of approaches for behavioral choice and design have arisen. The ultimate judge is the appropriateness of the robotic response to a given task and environment. Some methods currently used for specifying and designing robotic behaviors are described below.

1. Ethologically guided/constrained design. As previously mentioned, studies of animal behavior can provide powerful insights into the ways in which behaviors can be constructed. Roboticists can put models generated by biological scientists to good use. One such example comes from Arbib and House's (1987) studies of the navigational behavior of the toad and its relationship to Arkin's (1989a) schema-based robotic navigational system (section 4.4). In this instance, motion divergence fields are specified for a toad navigating amid a collection of poles toward a can of worms. This model provides an analogous means for representing robot behaviors using a modified potential (force) field method (figure 3.2). The key phrase for the design process here is "ethologically guided": consulting the biological literature for classifications, decompositions, and specifications of behaviors that would be useful for robotic systems, but not necessarily being constrained by them.

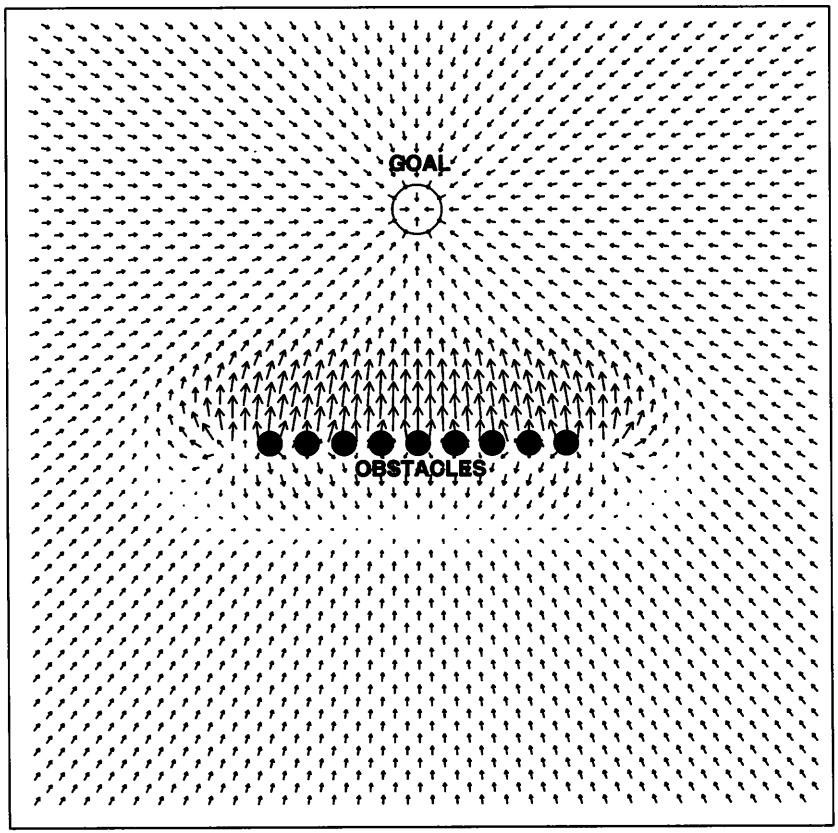
Other researchers, epitomized by Beer (Beer, Chiel, and Sterling 1990) (discussed earlier in section 2.5.3), look toward high-fidelity models of the neurological substrate of an animal (in Beer's case, the cockroach) in their attempt to emulate an appropriate behavioral response by a robot. These scientists choose to deliberately constrain their behavioral models to match those of the animal under study. In many ways this overconstrains the problem of producing intelligent behavior in a robot, but as a side effect this research can potentially answer interesting questions regarding actual biological behavior, for example in terms of predictive modeling.

The methodology for designing ethologically guided/constrained behaviors is illustrated schematically in figure 3.3. A model is provided from a scientific study, preferably with an active biological researcher in tow. The animal model is then modified as necessary to realize it computationally, and is then grounded within the robot's sensorimotor capabilities. The results from the robotic experiments are then compared to the results from the original biological studies, and either the biological model or its robotic alter ego are



(A)

Figure 3.2



(B)

Figure 3.2 ***(continued)***
Toad/robot navigational model (A) Represents a model of a toad's attraction to a can of worms, avoidance from a pole fence, and an egocentric animal potential for motion. The vectors represent the most likely direction of motion for the animal at each point in space. This model was shown to be consistent with experimentally observed animal data (after Arbib and House 1987). (B) Represents a set of robotic behaviors for similar circumstances: avoid-static-obstacles and move-to-goal. An egocentric potential is not needed in this similar, yet different, representation.

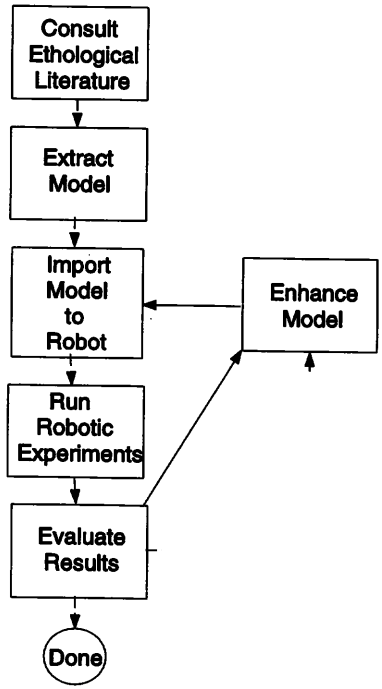


Figure 3.3
Design methodology for ethologically guided systems (dotted lines indicate optional pathway).

modified or enhanced in an attempt to produce results more in agreement with the original animal data. The results of these experiments have two customers: roboticists, who can use these insights to produce even more-intelligent machines, and experimental biologists, who can use them to develop and test their theories of animal behavior.

2. Situated activity–based design. Situated activity means that a robot's actions are predicated upon the situations in which it finds itself. Hence the perception problem is reduced to recognizing what situation(s) the robot is in and then choosing one action (out of perhaps many) to undertake. As soon as the robot finds itself in a new situation, it selects a new and more appropriate action. These manifold situations can be viewed as microbehaviors, that is, behaviors specified and useful only in very limited circumstances. Designing a robot based on this methodology requires a solid understanding of the relationship between the robotic agent and its environment. The design strategy typifying this method appears in figure 3.4.

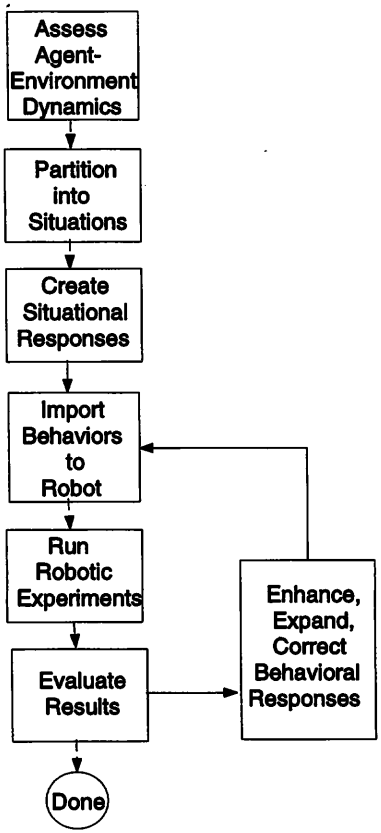


Figure 3.4
Situated activity design methodology.

Arbitrarily complex situations can be created and specified that may have no biological basis. Pengi (Agre and Chapman 1987) is a system that characterizes situations by their *indexical-functional* aspects. *Indexical* refers to what makes the circumstances unique, *functional* refers to a robotic agent's intended outcome or purpose in a given situation. This system uses lengthy phrases to characterize particular situations that demand certain responses. For example (paraphrasing the situations from the original Pengi somewhat), the-block-I-need-to-kick-at-the-enemy-is-behind-me is a situation that requires the agent to backtrack to obtain the object in question. I've-run-into-the-edge-of-the-wall requires that the robot turn and move along the edge. These situations can be highly artificial and arbitrarily large in number. Coordination in Pengi

is handled by an arbitration mechanism, where one of the candidate actions is chosen (there may be many applicable) and executed. Indeed, one candidate action may be in conflict with another. Hopefully the best action is chosen, but there is no guarantee, as no planning is conducted, nor does the mechanism project the consequences of undertaking any action.

Assuming that there is no limit to the number of situational conditions that can be enumerated leads us to a more expansive version of this theory of situated activity: universal plans. Universal plans, as developed by Schoppers (1987), require the robotic agent to have the ability of recognizing each unique situation for what it is and then selecting an appropriate action for each possible world state. These universal plans cover the entire domain of interaction, use sensing to conduct the classification, and presuppose no ordering on the situations or even the type of situations that might arise for that matter (Schoppers 1989). Sensing is conducted continuously, so situational assessment, and thus appropriate response, is continuously reevaluated.

To deal with the issue of the sheer bulk regarding the vast number of possible situations, the idea of caching plans is forwarded. Despite this technique, universal plans have encountered significant criticism (e.g., Ginsberg 1989) predominantly due to the immensity of the numbers of plans required and the potential irrelevancy of most. Even the harshest critics acknowledge that more limited versions of the situated activity paradigm have utility, even when it is designed to include not only routine situations but a wide range of contingent ones. The argument, however, that an enumeration of *every* possible situation (i.e., universal plans) is impractical at best and mathematically intractable at worst is a valid one.

Reactive action packages (RAPs) (Firby 1989) constitute an unusual variant on situation-driven execution. As with the other methods, the current situation provides an index into a set of actions regarding how to act in that environment. RAPs, however, operate at a coarser granularity than the other situated-action approaches and provide multiple methods of acting within a given context. RAPs consist of a set of methods specific to a task-situation, and for each of those methods, a sequence of steps to accomplish the task is provided (a kind of "sketchy" plan). RAPs differ from most reactive systems, however, in that they are not truly behavior based (but rather task based) and in that the system relies heavily on a strong explicit internal world model.

3. Experimentally driven design. Experimentally driven behaviors are invariably created in a bottom-up manner. The basic operating premise is to endow a robot with a limited set of capabilities (or competences), run experiments in the real world, see what works and what does not, debug imperfect behaviors, and

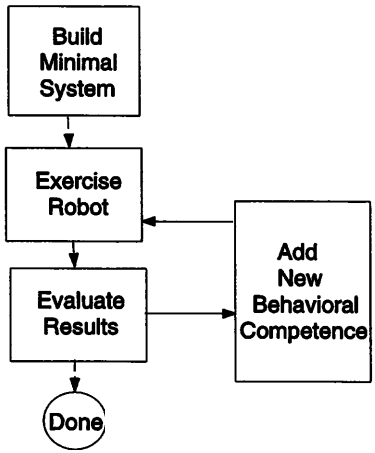
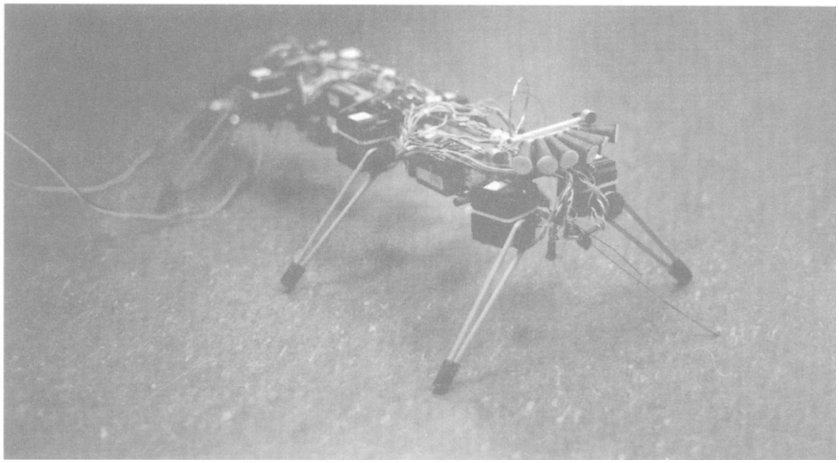


Figure 3.5
Experimentally driven methodology for behavioral design.

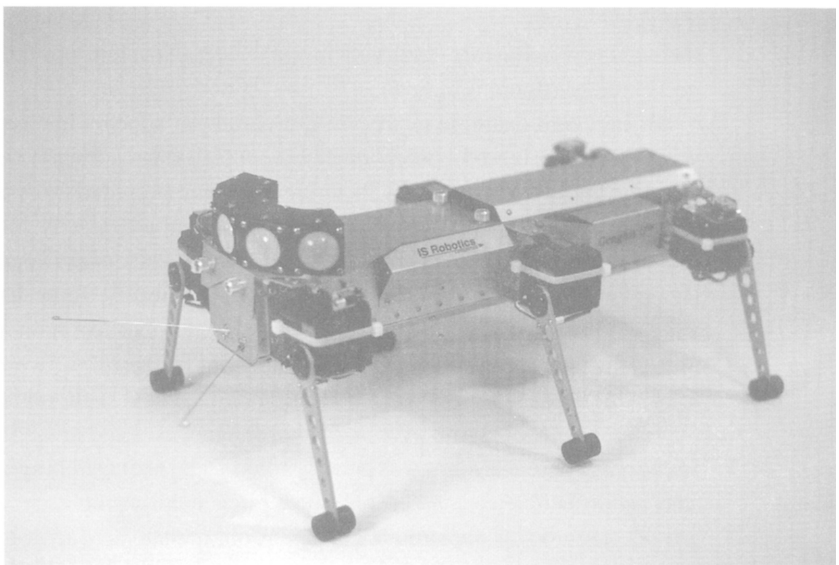
then add new behaviors iteratively until the overall system exhibits satisfactory performance (figure 3.5).

An excellent example of this design paradigm appears in Brooks' (1989a) work on the design of a behavior-based controller for a legged walking robot. Initially the robot (panel (A) in figure 3.6) was provided with the ability to stand up and conduct a simple walk. This worked adequately for smooth terrain but posed problems as the robot attempted to walk over irregular surfaces. Based on the requirements of this extended capability, force balancing was added to modify the leg controllers and help the robot maintain a steady posture. Whiskers (protruding sensors in the front of the robot) were then added to provide more warning to the control system to deal with larger objects that the robot needed to climb over. A final problem was noted involving balance in situations where the robot was heavily tilted fore or aft (pitching). To compensate, an inclinometer coupled with new pitch stabilization code was added to provide even better performance as the robot maneuvered over highly irregular terrain. It was then decided to allow the robot to track warm objects such as people, so infrared sensors were added, coupled with a new behavior to provide prowling. Each of these competencies was added incrementally, based upon the results of previous experiments and the goal of providing greater utility for the robotic system. Section 4.3 details the development of this system.

In Payton et al. 1992, fault tolerance is introduced into reactive robotic systems through the design of suitable behaviors that can handle unanticipated



(A)



(B)

Figure 3.6
(A) Original Genghis. (Photograph courtesy of Rodney Brooks.) (B) Genghis II—a robotic hexapod, commercial successor to the original Genghis. (Photograph courtesy of IS Robotics, Somerville, MA.)

contingencies as they arise. This design methodology, dubbed "Do whatever works," has a goal of generating a sufficiently general set of low-level behaviors that when activated can cope with events beyond the initial designer's vision. Redundancy is the key feature, that is, allowing things to be accomplished in more than one way and then designing a controller capable of selecting the most successful behavior for the current situation. In a sense, this feature is also embodied in the RAPs system described previously, in which multiple methods are used to accomplish a task.

Ferrell (1994) developed a complex control system for another walking hexapod potentially suitable as a lunar or Mars rover using this bottom-up strategy. In this implementation, 1,500 concurrent processes supported locomotion over rough terrain and provided the requisite sensing with a significant level of fault tolerance. The entire system was constructed without any reliance on simulation technology. Earlier, Connell (1989a) demonstrated the efficacy of this experimental method with the design of a mobile manipulator (figure 3.7). In particular, the arm controller for this system consisted of fifteen independent behaviors capable of finding a soda can, then grabbing, transporting, and depositing it at another location.

Whatever the design basis, a generic classification of robot behaviors can be used to categorize the different ways in which a robotic agent can interact with its world:

```
Exploration/directional behaviors (move in a general direction)
  heading based
  wandering
Goal-oriented appetitive behaviors (move towards an attractor)
  discrete object attractor
  area attractor
Aversive/protective behaviors (prevent collisions)
  avoid stationary objects
  elude moving objects  (dodge, escape)
  aggression
Path following behaviors (move on a designated path)
  road following
  hallway navigation
  stripe following
Postural behaviors
  balance
  stability
```

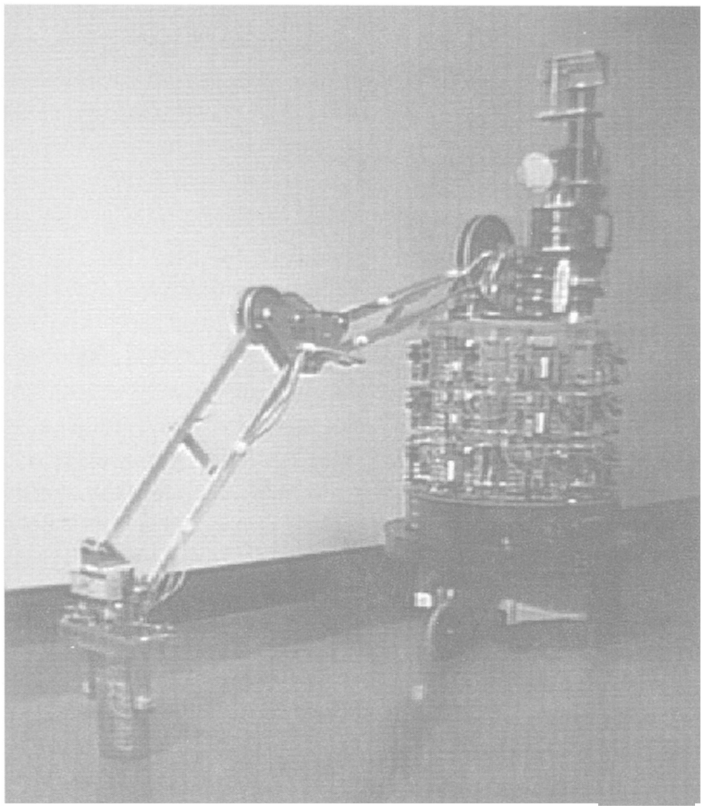


Figure 3.7
Herbert—a mobile manipulator. (Photograph courtesy of Jon Connell.)

```
Social/cooperative behaviors
  sharing
  foraging
  flocking/herding
Teleautonomous behaviors (coordinate with human operator)
  influence
  behavioral modification
Perceptual behaviors
  saccades
  visual search
  ocular reflexes
```

```
Walking behaviors (for legged robots)
  gait control
Manipulator-specific behaviors (for arm control)
  reaching
Gripper/dextrous hand behaviors (for object acquisition)
  grasping
  enveloping
```

Perceptual support is required to implement any of these behaviors. Chapter 7 describes how perception can be tailored to behavioral need.

3.2 EXPRESSION OF BEHAVIORS

Several methods are available for expressing robotic behavior. This book employs three: Stimulus-response (SR) diagrams, functional notation, and finite state acceptor (FSA) diagrams. These methods will be used throughout the text in representing various behavior-based systems. SR diagrams will be used for graphic representations of specific behavioral configurations, functional notation for clarity in design of the systems, and FSAs whenever temporal sequencing of behaviors is required.

3.2.1 Stimulus-Response Diagrams

Stimulus-response (SR) diagrams are the most intuitive and the least formal method of expression. Any behavior can be represented as a generated response to a given stimulus computed by a specific behavior. Figure 3.8 shows a simple SR diagram.

Figure 3.9 presents an appropriate SR diagram for our navigational example (section 3.1.2). Here five different behaviors are employed in the task of getting to the classroom. The outputs of each behavior is channeled into a coordination mechanism (schematized here) that produces an appropriate overall motor response for the robot at any point in time given the current existing environmental stimuli. Section 3.4 discusses further the problem and methods of coordinating behaviors.

Figure 3.8
SR diagram of a simple behavior.

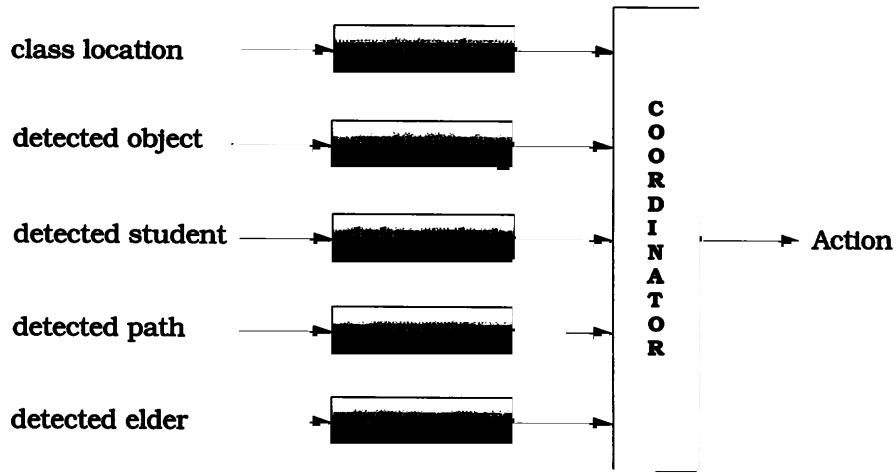


Figure 3.9
SR diagram for classroom navigation robot.

3.2.2 Functional Notation

Mathematical methods can be used to describe the same relationships using a functional notation,

$b(s) = r$,

meaning behavior b when given stimulus s yields response r. In a purely reactive system, time is not an argument of b, as the behavioral response is instantaneous and independent of the system's time history.

A functional expression of the behaviors necessary to carry out our example navigational task of getting to the classroom would appear as:

```
coordinate-behaviors[
                move-to-classroom(detect-classroom-location),
                avoid-objects(detect-objects),
                dodge-students(detect-students),
                stay-to-right-on-path(detect-path),
                defer-to-elders(detect-elders)
] = motor-response
```

The `= motor-response` is usually implicit and is generally not written when using this notation.

Each of the five behaviors listed can produce an output depending on the current environmental stimuli. A coordination function determines what to do with those outputs (e.g., selecting one of them or combining them in some meaningful way). It is not a trivial problem to ensure that the outputs of each behavioral function are in a form that can be coordinated, as section 3.4 will detail.

Coordinated functions can also be the arguments for other coordinating functions. For example

```
coordinate-behaviors[
            coordinate-behaviors(behavioral-set-1)
            coordinate-behaviors(behavioral-set-2)
            coordinate-behaviors(behavioral-set-3)
]
```

where each of the behavioral sets is a set of primitive behaviors. Clearly this notation readily permits a recursive formulation of behavior, ultimately grounded in physical robotic hardware but able to move upward into arbitrary levels of abstraction.

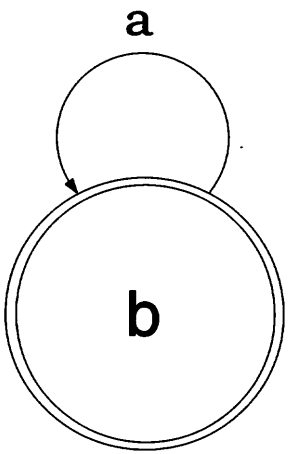
Functional notation has an interesting side effect in that it is fairly straightforward to convert this representation into a computer program. Often a functional programming language such as LISP is used, although the C language also enjoys widespread usage.

3.2.3 Finite State Acceptor Diagrams

Finite state acceptors (Arbib, Kfoury, and Moll 1981) have very useful properties when describing aggregations and sequences of behaviors (section 3.4.4). They make explicit the behaviors active at any given time and the transitions between them. They are less useful for encoding a single behavior, which results in a trivial FSA (figure 3.10).

In figure 3.10, the circle b denotes the state where behavior b is active and is accepting any stimulus input. The symbol a denotes *all* input in this case.

A finite state acceptor M can be specified by a quadruple (Q, δ, q_0, F) with Q representing the set of allowable behavioral states; δ being a transition function mapping the input and the current state to another, or even the same, state; q_0 denoting the starting behavioral configuration; and F representing a set of accepting states, a subset of Q, indicating completion of the sensorimotor task. δ can be represented in a tabular form where the arcs represent state transitions in the FSA and are invoked by arriving stimuli.



δ	q	*input*	$\delta(q, input)$
	b	a	b

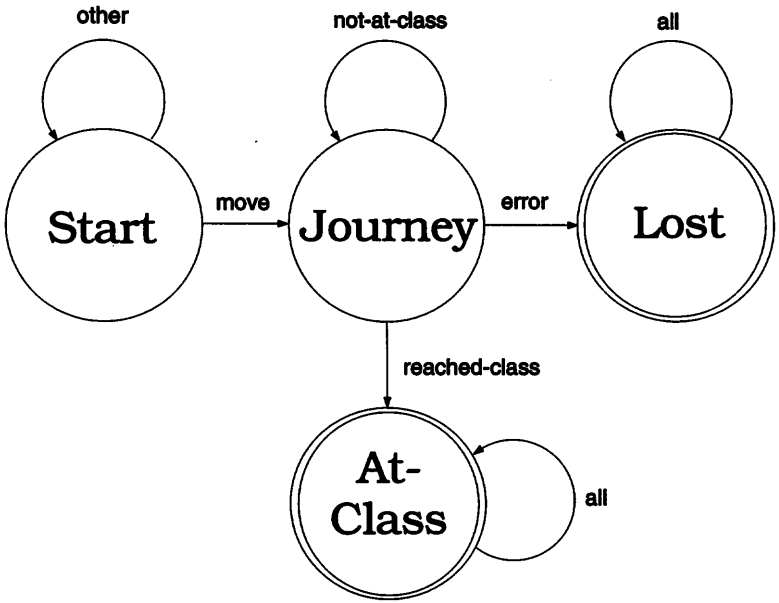
Figure 3.10
FSA for a simple behavior. The table shows the allowable states, the inputs that induce a transition from one state to another, and the resulting state. For this trivial FSA, all inputs result in the same state.

For the trivial example shown in figure 3.10:

$M = \{\{b\}, \delta, b, \{b\}\}$.

FSAs are best used to specify complex behavioral control systems where entire sets of primitive behaviors are swapped in and out of execution during the accomplishment of some high-level goal (Arkin and MacKenzie 1994). Gat and Dorais (1994) have also expressed the need for sequencing behaviors. FSAs provide a ready mechanism to express these relationships between various behavioral sets and have been widely used within robotics to express control systems. In their MELDOG system, Tachi and Komoriya (1985) use an automaton map to capture actions that should be executed at various places within the world. Similar examples exist of FSA usage for guiding vision-based robots (Fok and Kabuka 1991; Tsai and Chen 1986). Brooks (1986) has also used a variation, augmented finite state machines (AFSMs), to express behaviors within his subsumption architecture (section 4.3). In this text, however, we use the notation developed in Arkin and MacKenzie 1994 based on the formalisms described in Arbib, Kfoury, and Moll 1981.

The classroom navigation example can also be expressed with FSAs, although the result is of a decidedly different character (figure 3.11). This example has four different states: start, journey, lost, and at-class. The last two



δ	q	*input*	$\delta(q, input)$
	start	move	journey
	start	other	start
	journey	error	lost
	journey	not-at-class	journey
	journey	reached-class	at-class
	at-class	all	at-class
	lost	all	lost

Figure 3.11
FSA representing classroom navigation example.

are terminal states: lost is abnormal, at-class normal. Journey, the main behavioral state, actually consists of an assemblage (a coordinated collection) of the five other low-level behaviors mentioned earlier (move-to-classroom, avoid-objects, dodge-students, stay-to-right-on-path, and defer-to-elders). Specifically,

$M = \{\{start, journey, lost, at\text{-}class\}, \delta, start, \{lost, at\text{-}class\}\}$.

The FSA provides us with a higher level of abstraction by which we can express the relationships between sets of behaviors.

To further illustrate this, let's look at an even more complex example. Figure 3.12 depicts an FSA constructed for a robot used in a competition conducted by the American Association for Artificial Intelligence. Here, a collection of high-level behaviors, each represented schematically as a state, encodes the robot's goal of moving about an arena looking for ten distinct poles, then moving to each of those poles in sequence. This robot has three major behavioral states, wander, move-to-pole, and return-to-start. Move-to-pole consists of a subset of actions for selecting a pole, orienting the robot so it points toward the pole, moving to the pole, and tracking the pole visually during motion until it is reached. In this case,

$$M = \{\{start, find\text{-}next\text{-}pole, move\text{-}to\text{-}pole, wander, return\text{-}to\text{-}start, halt\}, \delta, start, \{halt\}\},$$

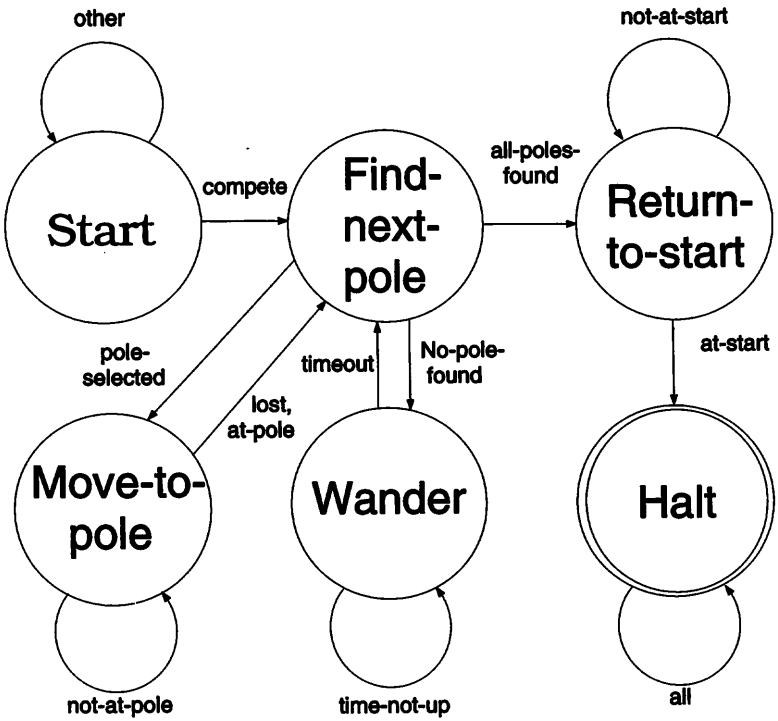
where δ contains the transition information depicted in figure 3.12. The FSA shows the sequencing between behaviors as the robot carries out its mission (figure 3.13). More details on this task and robot can be found in Arkin et al. 1993. Incidentally, the use of finite state descriptions in the University of Southern California's Phony Pony project on quadruped locomotion (McGhee 1967) is probably the first example of their application for specifying a robot control system.

3.2.4 Formal Methods

Formal models for behavior-based robotics can potentially provide a set of very useful properties to the robot programmer:

- They can be used to verify designer intentions.
- They can facilitate the automatic generation of robotic control systems.
- They provide a complete common language for the expression of robot behavior.
- They provide a framework for conducting formal analysis of a specific program's properties, adequacy, and/or completeness.
- They provide support for high-level programming language design.

Several formal methods have been developed for specifying and designing behavior-based robotic systems. A brief review of two representative strategies is presented below.



δ	q	*input*	$\delta(q, input)$
	start	compete	find-next-pole
	start	other	start
	find-next-pole	all-poles-found	return-to-start
	find-next-pole	no-pole-found	wander
	find-next-pole	pole-selected	move-to-pole
	return-to-start	not-at-start	return-to-start
	return-to-start	at-start	halt
	move-to-pole	lost	find-next-pole
	move-to-pole	at-pole	find-next-pole
	move-to-pole	not-at-pole	move-to-pole
	wander	time-not-up	wander
	wander	timeout	find-next-pole
	halt	all	halt

Figure 3.12
FSA representing robot competition example.

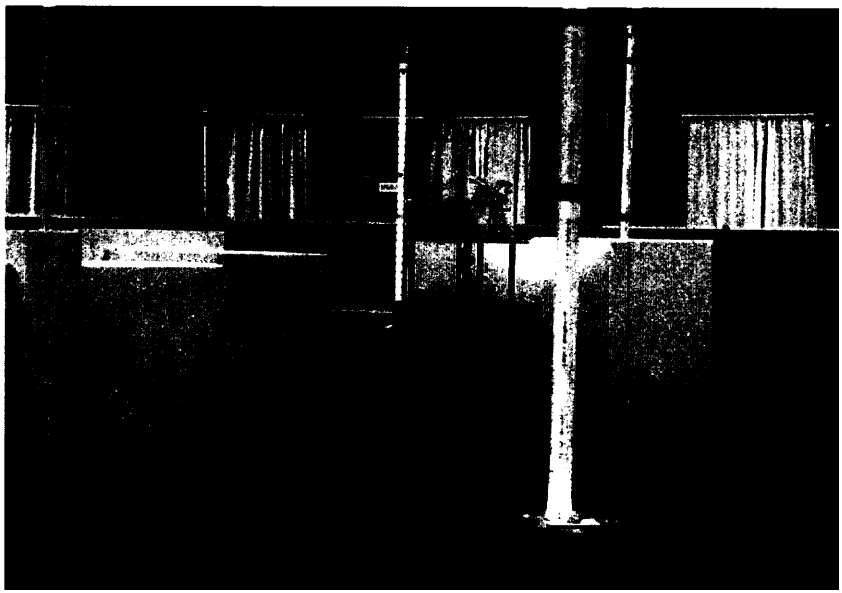


Figure 3.13
Robot executing behaviors at competition arena.

3.2.4.1 *RS*

Lyons and Arbib (1989) developed the *RS* (robot schema) model as a method for expressing distributed sensor-driven robot control programs. A process algebra is used that permits the composition of a network of processes called *schemas* (behaviors). Process composition operators have been defined as a basis for creating these networks, which include methods for conditional, sequential, parallel, and iterative structures. Preconditions are established for coordination operators to ensure a smooth flow of control during execution.

In particular, a port automata model has been adopted as the underpinning for expressing the relationships between schemas. Port automata can be viewed as an extension of FSAs with supplemental formal methods for specifying the interconnections between states. Schemas communicate with each other via predefined input-output ports using synchronous message passing techniques. A schema's behavioral description, which encodes its response to any input messages, fully determines its action. Schemas are aggregated via a nesting mechanism termed an assemblage. Each assemblage recursively encodes a unique network of schemas or other assemblages.

A high-level algebraic *RS* encoding for the navigational example used throughout this chapter would be:

```
Class-going-robot = (Start-up ; (done? , Journey) : At-classroom)
Journey = (move-to-classroom , avoid-objects, dodge-students,
           stay-to-right-on-path, defer-to-elders)
```

Translating, the `Class-going-robot` consists of a robot that, beginning from an initial start-up state, sequentially transitions to the `Journey` state (the sequential operator is denoted ;) and which then remains in the `Journey` state with a concurrent monitor process checking for arrival at the classroom (the concurrency operator is denoted with a ,). If the robot is at the classroom it transitions to the `At-classroom` state (denoted by the conditional operator :). `Journey` consists of the concurrent execution of the behaviors specified during travel from one location to the next. This method combines advantages of both the functional and FSA methods into a single syntax.

There is far more to the *RS* model than what is presented here. The interested reader is referred to Lyons and Arbib 1989, Lyons and Hendriks 1994, and Lyons 1992 for more details.

3.2.4.2 Situated Automata

The situated automata model, developed by Kaelbling and Rosenschein (1991), recognizes the fundamental relationship an agent has as a participant within its environment. The model employs logical formalisms as underpinnings for the design of circuitry that corresponds to a robot's goals and intentions. The use of logic enables reasoning over the system, which can lead to the establishment of provable properties (Rosenschein and Kaelbling 1987), an important goal for the designer of any type of system. Rex (Kaelbling 1986), a LISP-based system, was the first language to embody the basic tools to generate specifications for synchronous digital circuitry embodying a reactive control program. Gapps is a more recently developed language that enables goals to be specified more directly and is inherently easier to use, akin to a higher generation language in conventional programming. Goals are either achieved, executed or maintained: Achieved goals are those that should be eventually realized; executed goals are those that should be done now; and maintained goals are those that should be preserved, as they have already been attained. Operators are defined in a LISP-like format that correspond to these three goal states: `(ach goal)` for achieve, `(do goal)` for execute, and `(maint goal)` for

maintain. The logical boolean operators `and`, `or`, `not`, `if` are used to create higher-level goals.

Circuits are generated from these high-level goal expressions. Standard logical methods can be used to compile the high-level circuitry into a collection of digital logic gates. This type of formalism provides a very concrete grounding onto actual digital hardware for creating situated automata robots.

A simplified Gapps specification for our ongoing classroom navigation example would be:

```
(defgoalr (ach in-classroom)
          (if (not start-up)
              (maint (and (maint move-to-classroom)
                          (maint avoid-objects)
                          (maint dodge-students)
                          (maint stay-to-right-on-path)
                          (maint defer-to-elders)
                     )
              )
           )
)
```

This encoding states that the robot is to achieve the goal of being in the classroom. If the robot is not in `start-up` state, then it is to journey to the location by maintaining the concurrent goals of moving to the classroom, avoiding objects, dodging students, staying to the right, and deferring to elders. Methods also exist to prioritize goals within the Gapps language should the need arise. As in the case with *RS,* there is far more to the Gapps language than can be discussed here, so the interested reader is referred to Kaelbling and Rosenschein 1991 for additional information.

Gapps circuitry was used for an unmanned underwater vehicle (UUV), described in Bonasso 1992. The basic goals established for this system were

- `(maint not-crashed)` which established the subgoal of `(ach avoid nearest obstacle)` along with other behaviors such as avoiding collision with the ocean bottom.
- `(ach wander)` endowed the robot with exploration capabilities.
- `(ach joystick goal-point)` allowed user directed input to control the direction of the robot.

The first two of these behaviors provided the UUV with the ability to navigate safely in a water tank. Additional behavioral goals were described us-

ing Gapps and tested in simulation, including `(maint best-heading)` and mission-specific tasks such as `(ach record-thermal-vent-event)` and `(ach quiescence)`.

3.3 BEHAVIORAL ENCODING

To encode the behavioral response that the stimulus should evoke, we must create a functional mapping from the stimulus plane to the motor plane. We will not concern ourselves at this point as to whether a behavioral response is appropriate to a given stimulus, only how to encode it.

An understanding of the dimensionality of a robotic motor response is necessary in order to map the stimulus onto it. It will serve us well to factor the robot's motor response into two orthogonal components: strength and orientation.

- *Strength* denotes the magnitude of the response, which may or may not be related to the strength of a given stimulus. For example, it may manifest itself in terms of speed or force. Indeed the strength may be entirely independent of the strength of the stimulus yet modulated by exogenous factors such as intention (what the robot's internal goals are) and habituation (how often the stimulus has been previously presented). We will later see that by controlling the strength of the response to a given stimulus, inroads are created for integrating goal-oriented planning into behavioral systems (chapter 6) as well as introducing the opportunity for adaptive learning methods (chapter 8).
- *Orientation* denotes the direction of action for the response, (e.g., moving away from an aversive stimulus, moving towards an attractor). The realization of this directional component of the response requires knowledge of the robot's kinematics. It may or may not be dependent on the stimulus's strength.

> In general, *kinematics* is the science of objects in motion, including aspects of position, velocity, and acceleration. This includes, in particular, all of the robot's geometric and time-based physical properties. *Dynamics* extends kinematics to include the study of the forces that produce motion in objects.

A behavior can be expressed as a triple (S, R, β) where S denotes the domain of all interpretable stimuli, R denotes the range of possible responses, and β denotes the mapping $\beta: S \rightarrow R$.

R—Range of Responses

Refining this further, the *instantaneous* response $\mathbf{r}$ (where $\mathbf{r} \epsilon R$) of a behavior-based reactive system can be expressed as a six-dimensional vector consisting of six subcomponent vectors. Each of the subcomponent vectors encodes the magnitude of the translational and orientational responses for each of the six degrees of freedom of motion of a general mobile robot.

> A *degree of freedom* (DOF) refers to one of the set of independent position variables, with respect to a frame of reference, necessary to specify an object's position within the world.

An unconstrained rigid object has six DOFs,

$$\mathbf{r} = [\mathbf{x}, \mathbf{y}, \mathbf{z}, \theta, \phi, \psi],$$

where the first three components of $\mathbf{r}$ represent the three translational degrees of freedom (x, y, z in three-dimensional cartesian coordinates), and the last three components encode the three rotational degrees of freedom (θ for roll, ϕ for pitch, ψ for yaw). Often pitch is alternatively referred to as tilt, and yaw as pan, as in a pan-tilt device. This is especially true in the context of controlling the pointing of sensors such as cameras.

For ground-based mobile robots, the dimensionality is often considerably less than six DOFs. For example, a robot that moves on flat ground and can rotate only about its central axis has only three degrees of freedom, $\mathbf{r} = [x, y, \theta]$, representing translation in the cartesian ground plane $[x, y]$ and the one degree of rotation θ (yaw, or alternatively pan).

Another factor that can limit the realization of a generated behavioral response is the robot's non-holonomicity.

> A *non-holonomic* robot has restrictions in the way it can move, typically because of kinematic or dynamic constraints on the robot, such as limited turning abilities or momentum at high velocities.

A truly holonomic robot can be treated as a massless point capable of moving in any direction instantaneously. Obviously this is a very strong assumption and does not hold for any real robot (although it is easy to make this assump-

tion in simulation, potentially generating misleading results regarding a control algorithm's utility on an actual robot). Omnidirectional robots moving at slow translational velocities, however, (that is, robots that can essentially turn on a dime and head in any direction), can often pragmatically be considered to be holonomic, but they are not in the strictest sense. Non-holonomicity is generally of great significance when there are steering angle constraints, such as in a car attempting to park parallel. If the wheels were capable of turning perpendicular to the curb, parking would be much easier, but as they cannot, a sequence of more complex motions is required to move the vehicle to its desired location.

The constraints imposed by non-holonomic systems can be dealt with either during the generation of the response, by including them within the function β, or after $\mathbf{r}$ has been computed, translating the desired response to be within the limitations of the robot itself.

S—The Stimulus Domain

S consists of the domain of all perceivable stimuli. Each individual stimulus or percept $\mathbf{s}$ (where $\mathbf{s} \epsilon S$) is represented as a binary tuple (p, λ) having both a particular type or perceptual class p and a property of strength λ. The complete set of all p over the domain S defines all the perceptual entities a robot can distinguish, that is, those things it was designed to perceive. This concept is loosely related to affordances as discussed in section 2.3. The stimulus strength λ can be defined in a variety of ways: discrete (e.g., binary: absent or present; categorical: absent, weak, medium, strong) or real valued and continuous.

In other words, it is not required that the mere presence of a given stimulus be sufficient to produce an action by the robot.

> The presence of a stimulus is necessary but not sufficient to evoke a motor response in a behavior-based robot.

We define τ as a threshold value, for a given perceptual class p, above which a response is generated.

Often the strength of the input stimulus (λ) will determine whether or not to respond and the magnitude of the response, although other exogenous factors can influence this (e.g., habituation, inhibition, etc.), possibly by altering the value of τ. In any case, if λ is nonzero, the stimulus specified by p is present to some degree.

Certain stimuli may be important to a behavior-based system in ways other than provoking a motor response. In particular, they may have useful side effects on the robot, such as inducing a change in a behavioral configuration even if they do not necessarily induce motion. Stimuli with this property are referred to as *perceptual triggers* and are specified in the same manner as previously described (p, λ). Here, however, when p is sufficiently strong, the desired behavioral side effect is produced rather than motion. We return to our discussion of stimuli in the context of perception in chapter 7.

β—The Behavioral Mapping

Finally, for each individual active behavior we can formally establish the mapping between the stimulus domain and response range that defines a behavioral function β where

$\beta(\mathbf{s}) \mapsto \mathbf{r}$.

β can be defined arbitrarily, but it must be defined over all relevant p in S. Where a specific stimulus threshold, τ, must be exceeded before a response is produced for a specific $\mathbf{s} = (p, \lambda)$:

β: $(p, \lambda) \mapsto$ {for all $\lambda < \tau$ then $\mathbf{r} = [0, 0, 0, 0, 0, 0]$ * no response *\
else $\mathbf{r}$ = arbitrary function} * response *\

where [0, 0, 0, 0, 0, 0] indicates that no response is required given the current stimuli **s**.

Examples

Consider the example of collision avoidance behavior. If an obstacle stimulus is sufficiently far away (hence weak), no actual action may be taken despite its presence. Once the stimulus is sufficiently strong (in this case measured by proximity), evasive action will be taken. To illustrate intuitively, imagine you are walking on a long sidewalk and you see someone approaching far ahead of you. In general, you would not immediately alter your path to avoid a collision, but rather you would not react to the stimulus until action is truly warranted. This makes sense, as the situation may change significantly by the time the oncoming walker reaches you. She may have moved out of the way or even turned by herself off the path, requiring no action whatsoever on your part.

The functional mapping between the strength of stimulus and the magnitude (strength) and direction of robotic motor response defines the design space for a particular robotic behavior. Figure 3.14 depicts two possible stimulus strength-response strength plots for the situation described above. One is a step

function in which, once the distance threshold is exceeded, the action is taken at maximum strength. The other formulation involves increasing the response strength linearly over some range of stimulus strength (measured by distance here). Other functions are of course possible. The response's orientation may also vary depending on how the behavior has been constructed. For example, the motor response may be directly away from the detected object (strict repulsion—move away), or alternatively tangential to it (circumnavigation—go left or right) (figure 3.15).

Associated with a particular behavior, β, may be a scalar gain value g (strength multiplier) further modifying the magnitude of the overall response $\mathbf{r}$ for a given $\mathbf{s}$:

$$\mathbf{r}' = g\mathbf{r}$$

These gain values are used to compose multiple behaviors by specifying their strengths relative to one another (section 3.4). In the extreme case, g can be used to turn off a behavior by setting it to 0, thus reducing $\mathbf{r}'$ to 0.

The behavioral mappings, β, of stimuli onto responses fall into three general categories:

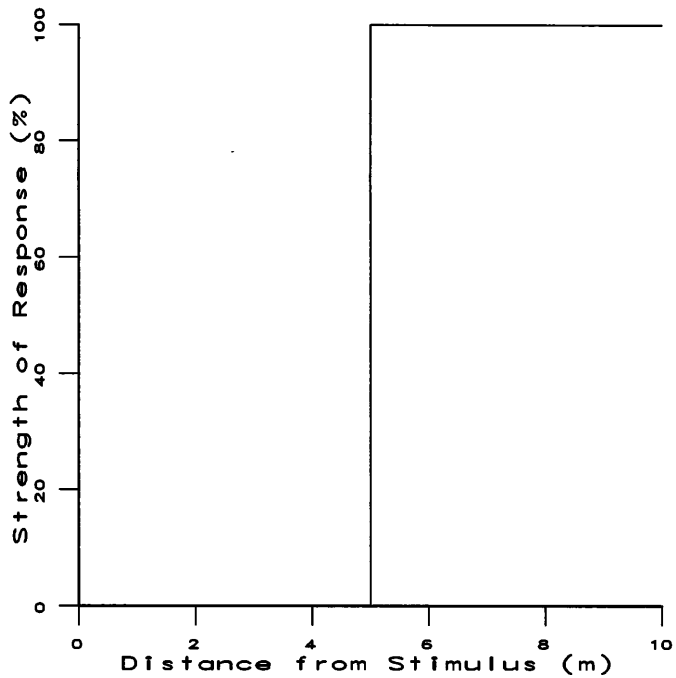
- Null: The stimulus produces no motor response.
- Discrete: The stimulus produces a response from an enumerable set of prescribed choices (all possible responses consist of a predefined cardinal set of actions that the robot can enact, e.g., turn-right, go-straight, stop, travel-at-speed-5). R consists of a bounded set of *stereotypical* responses enumerated for the stimulus domain S and specified by β.
- Continuous: The stimulus domain produces a motor response that is continuous over R's range. (Specific stimuli $\mathbf{s}$ are mapped into an infinite set of response encodings by β.)

Obviously it is easy to handle the null case as discussed earlier: For all $\mathbf{s}$, $\beta: \mathbf{s} \mapsto \mathbf{0}$. Although this is trivial, there are instances (perceptual triggers) where this response is wholly appropriate and useful, enabling us to define perceptual processes independent of direct motor action.

The methods for encoding discrete and continuous responses are discussed in turn.

3.3.1 Discrete Encoding

We have already seen one instance of discrete encodings, in section 3.1.3: situated action. For these cases, β consists of a finite set of (situation, response)



(A)

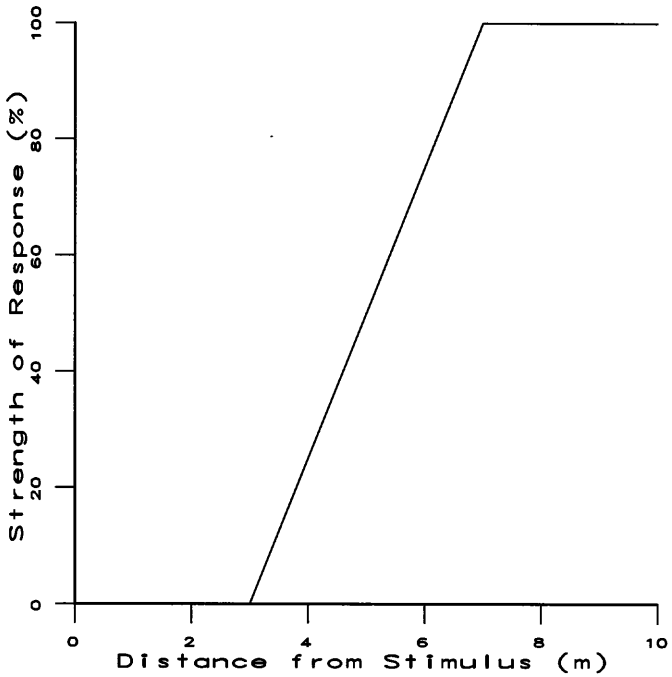
Figure 3.14

pairs. Sensing provides the index for finding the appropriate situation. The responses generated for a situation can be very simple, such as halt, or more complex, potentially generating a sequence of actions akin to the fixed-action patterns described in chapter 2.

Another strategy using discrete encodings involves the use of rule-based systems. Here β is represented as a collection of If-then rules. These rules take the general form:

IF *antecedent* **THEN** *consequent*

where the antecedent consists of a list of preconditions that must be satisfied in order for the rule to be applicable and the consequent contains the motor response. The discrete set of possible responses corresponds to the set of rules in the system. More than one rule may be applicable for any given situation. The strategy used to deal with *conflict resolution* typically selects one of the potentially many rules to use based on some evaluation function. Many rule-



(B)

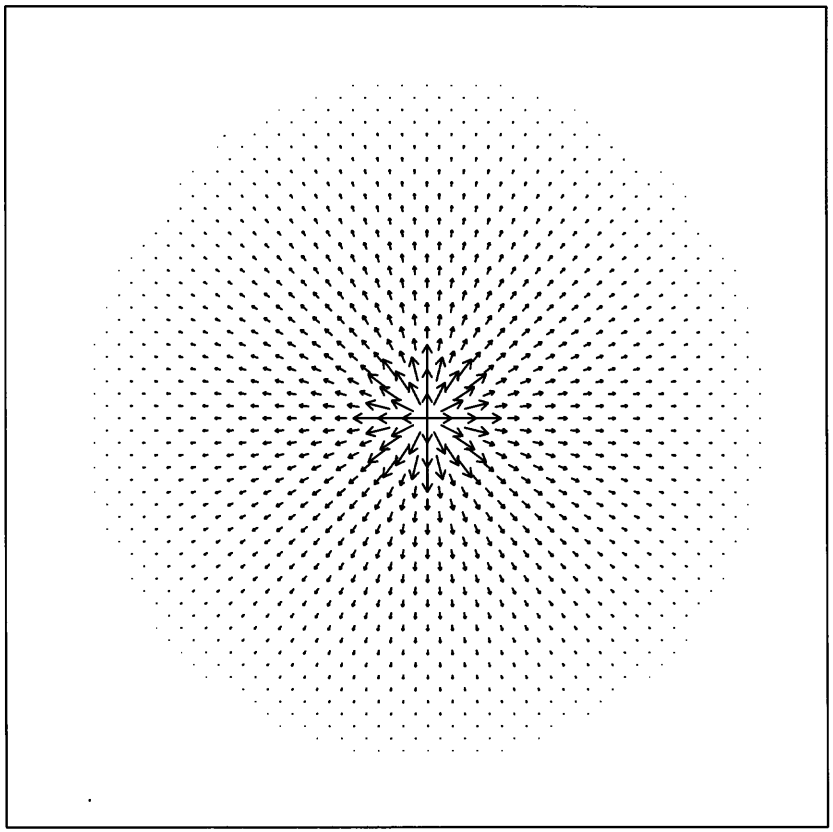
Figure 3.14 ***(continued)***
Stimulus distance/response strength plot: (A) step function, and (B) linear increase.

based behavioral systems encode their behaviors using fuzzy rules; we will study these systems in more detail in chapter 8.

Gapps (section 3.2.4.2) uses goal-reduction rules to encode the actions required to accomplish a task. Here the antecedent specifies a higher-level goal that if necessary will require that certain subgoals be achieved. Eventually these subgoals translate into specific motor commands (or action vectors, to use their terminology). One example for an underwater robot (Bonasso 1992) used the following Gapps rule:

```
(defgoalr (ach wander)
   (if (not (RPV-at-wander-angle))
       (ach turn to wander angle)
       (ach wander set point)))
```

Here the wander behavior requires that the robot turn to a new wander angle and then move to a newly established set point.

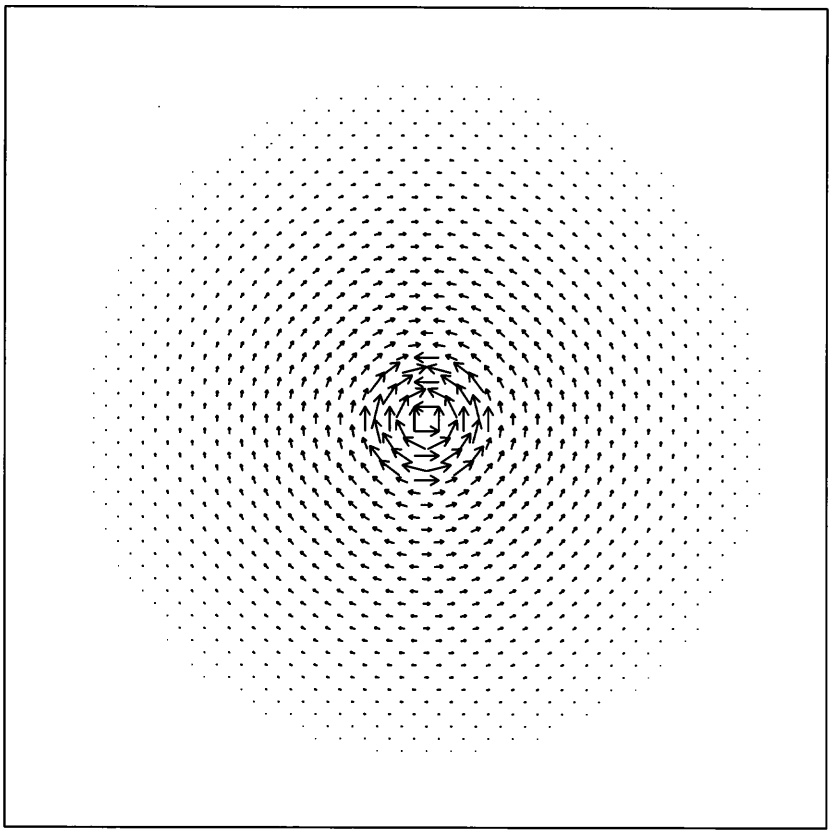


(A)

Figure 3.15

In Nilsson 1994, condition-action production rules control the robot. The condition (antecedent) is based partially on sensory information, whereas the consequent encodes a response for the robot to enact. The resulting actions are themselves *durative* instead of discrete, here meaning that the action resulting from the invocation of the rule persists indefinitely. (Contrast the durative response of "Move at a speed of five meters/second" with the discrete response "Move fifteen meters.") This *teleo-reactive* method is somewhat related to the circuitry models epitomized by Gapps, yet uses distinguishing formalisms that can potentially facilitate analysis.

Another example of discrete encoding involves rules encoded for use within the subsumption architecture (Brooks 1986). To facilitate the use of this partic-



(B)

Figure 3.15 ***(continued)***
Two directional encodings for collision avoidance: (A) repulsive, and (B) circumnavigation, with approach from the bottom of the figure.

ular approach to reactive systems, the Behavior Language was created (Brooks 1990a). This language embodies a real-time, rule-based approach for specifying and encoding behavior. Using a LISP-like syntax, rules are specified with the whenever clause:

(**whenever** *condition &rest body-forms*)

where the *condition* portion of the rule corresponds to the antecedent and the *&rest* to the consequent. Behaviors are constructed by collecting a set of these real-time rules into a group typically using the **defbehavior** clause:

(**defbehavior** *name &key inputs outputs declarations processes*)

where the list of rules appears in the *processes* location.

Similar rules were used for the control of a wheelchair in Connell and Viola 1990. A loose translation of a few of these behavioral rules include

- Approach: IF an object is detected beyond specified sonar range, THEN go forward.
- Retreat: IF an object is nearby according to sonar, THEN move backward.
- Stymie: IF all the front infrared sensors indicate an object immediately in front, THEN turn left.

The entire robotic system consisted of fifteen such behaviors encoding a mapping of direct sensing to motor activity. (These behaviors still require coordination, which is discussed in section 3.4).

3.3.2 Continuous Functional Encoding

Continuous response allows a robot to have an infinite space of potential reactions to its world. Instead of having an enumerated set of responses that discretizes the way in which the robot can move (e.g., {*forward*, *backward*, *left*, *right*, *speedup*, *slowdown*, . . . , *etc.*}), a mathematical function transforms the sensory input into a behavioral reaction. One of the most common methods for implementing continuous response is *based* on a technique referred to as the potential fields method. (We will revisit why the word *based* is italicized in the previous sentence after we understand the potential fields method in more detail.)

Khatib (1985) and Krogh (1984) developed the potential fields methodology as a basis for generating smooth trajectories for both mobile and manipulator robotic systems. This method generates a field representing a navigational space based on an arbitrary potential function. The classic function used is that of Coulomb's electrostatic attraction, or analogously, the law of universal gravitation, where the potential force drops off with the square of the distance between the robot and objects within its environment. Goals are treated as attractors and obstacles are treated as repulsors. Separate fields are constructed, based upon potential functions, to represent the relationship between the robot and each of the objects within the robot's sensory range. These fields are then combined, typically through superpositioning, to yield a single global field. For path planning, a smooth trajectory can then be computed based upon the gradient within the globally computed field. A detailed presentation of the potential fields method appears in Latombe 1991.

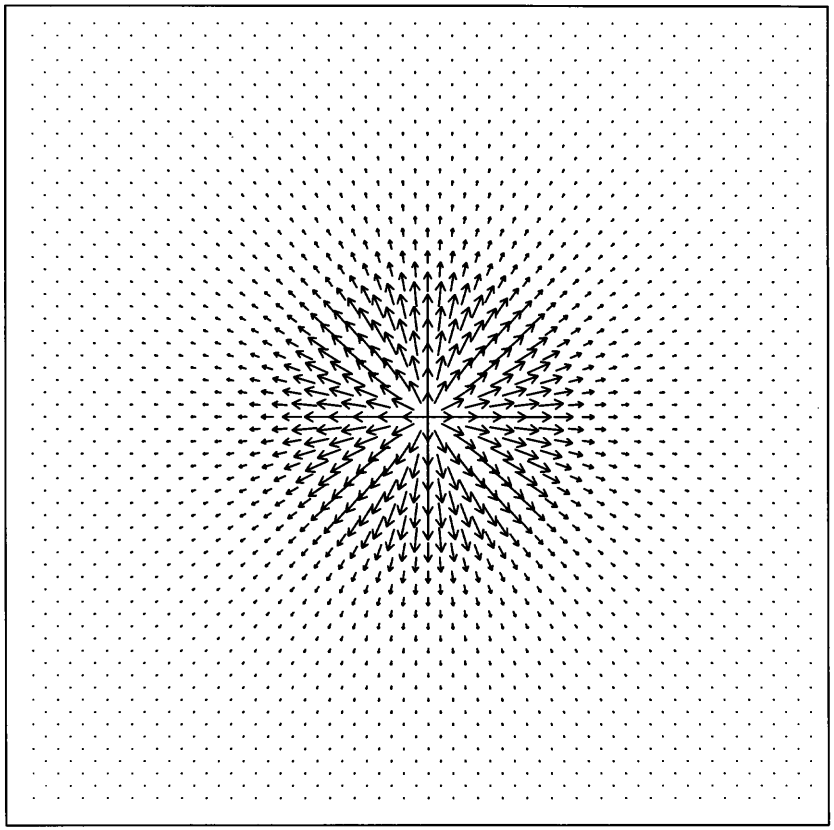
Using the inverse-square law expressing the relationship between force and distance,

$$Force \propto \frac{1}{Distance^2}$$

we construct a field for a repulsive obstacle, shown in (A) in figure 3.16. A ballistic goal attraction field, where the magnitude of attraction is constant throughout space, is depicted for an attractor located in the lower right of (B) in figure 3.16. In figure 3.17, (A) shows the linear superposition of these two fields, with (B) illustrating an example trajectory for a robot moving within this simple world.

Because potential fields encode a continuous navigational space through the sensed world (i.e., the force can be computed at any location), they provide an infinite set of possibilities for reaction. Potential fields are not without their problems, however, (Koren and Borenstein 1991). In particular, they are vulnerable to local minima (locations where the robot may get stuck) or cyclic-oscillatory behavior. Numerous methods have been developed to address these problems, including the use of harmonic potential fields (Kim and Khosla 1992; Connolly and Grupen 1993), time-varying potential fields (Tianmiao and Bo 1992), random noise injected into the field (Arkin 1987a) and adaptive methods (Clark, Arkin, and Ram 1992), among others. We will examine more closely the methods for combining potential fields in section 3.4.

Another seemingly significant problem with the use of the potential fields method is the amount of time required to compute the entire field. Reactive robotic systems eliminate this problem by computing each field's contribution at the instantaneous position merely where the robot is currently located (Arkin 1989b). This is why we said earlier that these techniques are only *based* on the potential fields method: no field is computed at all. No path planning is conducted at all: Rather, the robot's reaction to its environment is recomputed as fast as sensory processing will permit. One of the major misconceptions in understanding reactive methods based on potential fields is a failure to recognize the fact that the only computation needed is that required to assess the forces from the robot's current position within the world. This method is thus inherently very fast to compute as well as highly parallelizable. When the entire field is represented in a figure, it is only for the reader's edification. Reiterating, behavior-based reactive systems using potential fields do not generate plans based on the entire field but instead react only to the robot's egocentric perceptions of the world. This is of particular importance when the world is dynamic (i.e., there are moving objects, thus invalidating static planning tech-

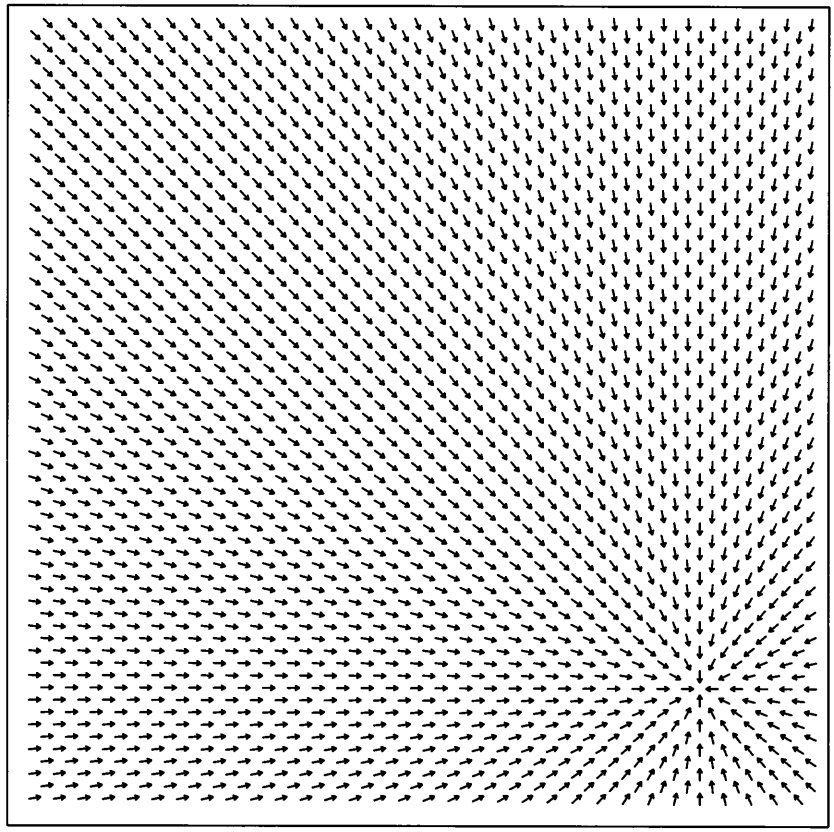


(A)

Figure 3.16

niques) or sensor readings are noisy or unreliable (as a plan generated on bad data is likely to be defective itself). Frequent resampling of the world helps overcome many of these deficiences.

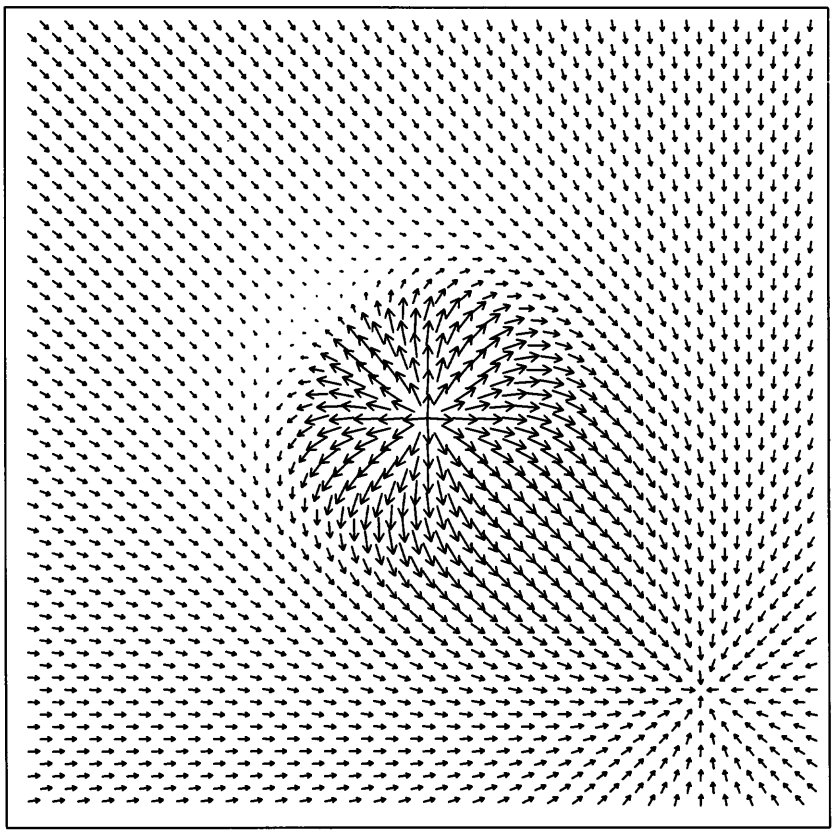
Slack (1990) developed another method for describing continuous pathways through navigational space. Navigational templates (NATs) are defined as arbitrary functions to characterize navigational space that do not necessarily have any correlation with typical potential fields methods. These NAT primitives rather characterize space on a task-oriented approach and are defined on an as-needed basis. An example of this method involves the use of spin-based techniques for obstacle treatment, circumventing the problem of local min-



(B)

Figure 3.16 ***(continued)***
Potential fields for (A) an obstacle and (B) a goal located in the lower right side of the figure.

ima found in traditional potential fields methods. In figure 3.18, (A) shows a spin-based version for obstacle avoidance for the situation depicted in (B) in figure 3.16. The results of superpositioning this new template (as opposed to field) with (A) in Figure 3.16 are shown in (B) in Figure 3.18. By using knowledge of the goal's location relative to the detected obstacle, a spin direction is chosen, either clockwise (when the obstacle is to the right of the goal, viewed from above) or counterclockwise (when it is to the left). Unfortunately, the modularity of the behavior is somewhat compromised with this technique, for

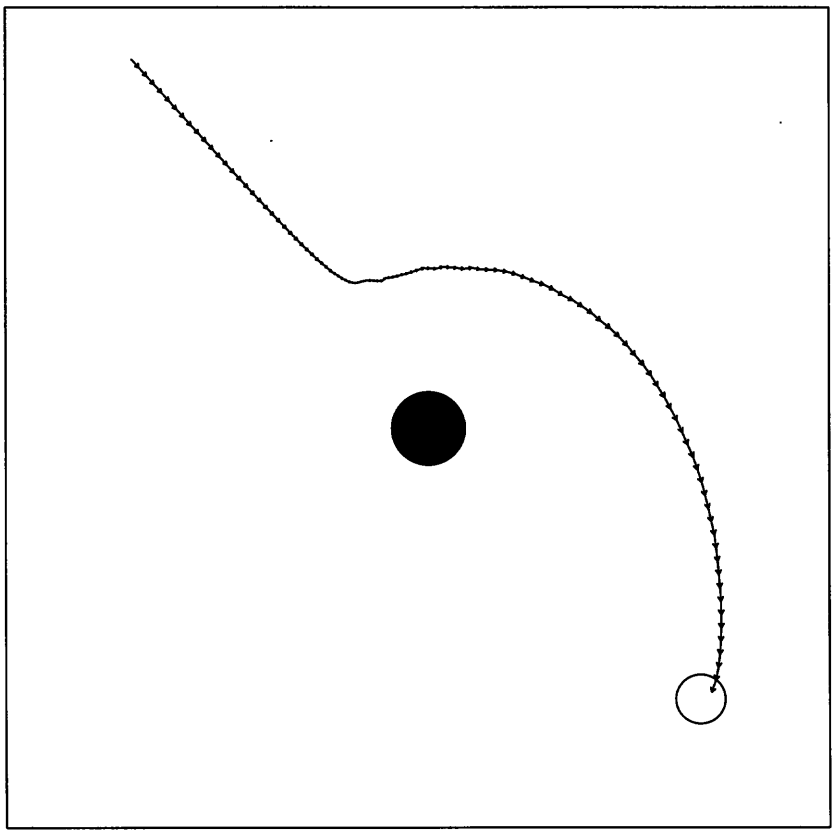


(A)

Figure 3.17

the correct choice for the obstacle's spin direction requires knowledge of the goal location, which normally is used only within the goal attraction behavior.

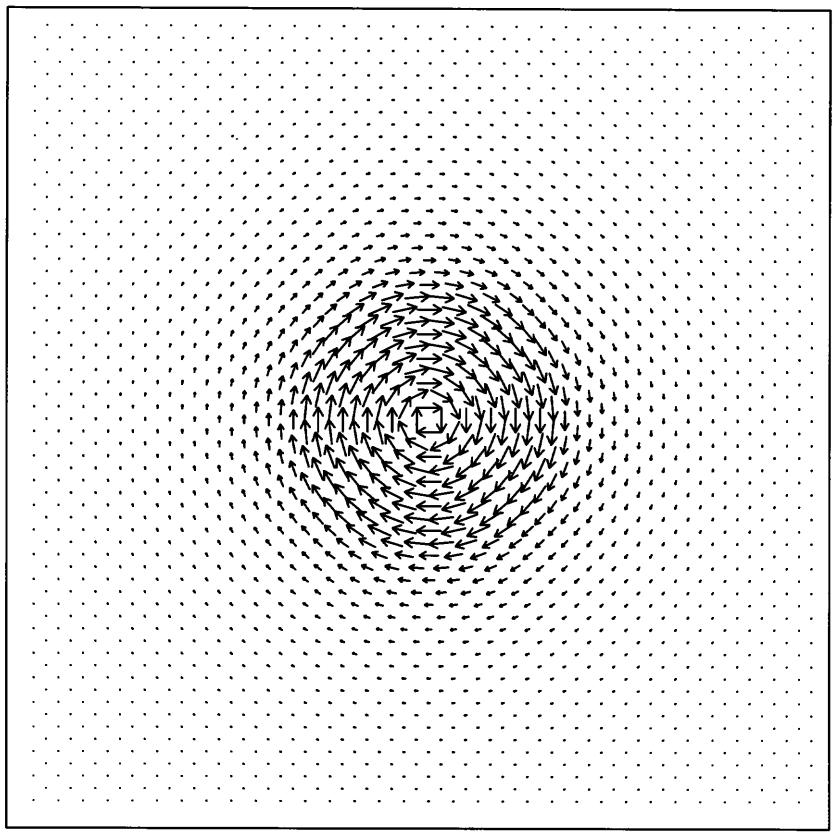
One other novel representation uses a technique called deformation zones for collision avoidance (Zapata et al. 1991). This technique defines two manifolds: the information-boundary manifold, which represents the maximum extent of sensing, and the range manifold, constructed from the readings of all active sensors, which can be viewed as a deformed version of the information-boundary manifold because of the presence of obstacles (figure 3.19). Perception produces deformation of the information-boundary manifold, and control strategies are generated in response to remove the deformation. In the example



(B)

Figure 3.17 ***(continued)***
(A) Linear superposition of fields in figure 3.16. (B) An example trajectory through this navigational space.

shown, the robot would steer to the left. In this approach, reactive behaviors are defined, including emergency stop, dynamic collision avoidance, displacement (orientation), and target following. One interesting aspect of this work allows for the definition of a variable collision avoidance zone dependent on the robot's velocity (i.e., when the robot is moving faster it projects further into the information-boundary manifold, and when it moves more slowly the information-boundary manifold shrinks). This enables effective control of the robot's speed by recognizing that if the deformation zone cannot be restored by steering, it can be restored by slowing the vehicle down.

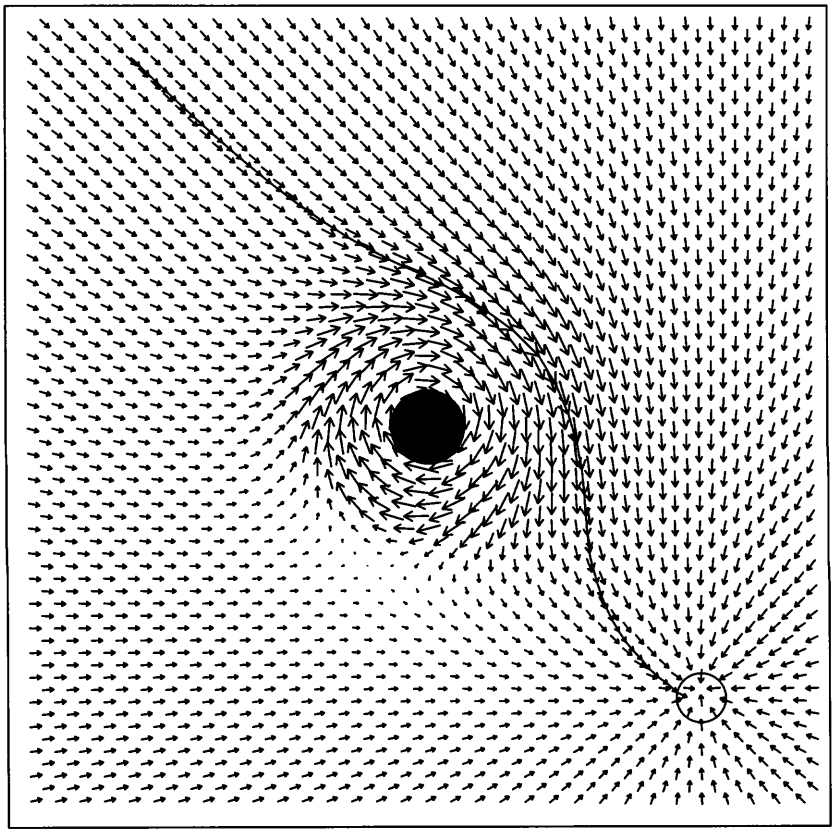


(A)

Figure 3.18

3.4 ASSEMBLING BEHAVIORS

Having discussed methods to describe individual behaviors, we now study methods for constructing systems consisting of multiple behaviors. This study requires the introduction of notational formalisms that we will use throughout this book and an understanding of the different methods for combining and coordinating multiple behavioral activity streams. We first examine, however, the somewhat controversial notion of emergent behavior.



(B)

Figure 3.18 ***(continued)***
(A) NAT spin field for the obstacle in (B) in figure 3.16. (B) Superposition of fields (A) in this figure and figure 3.16b. Also superimposed on this figure is the path for a robot using this combined field for the same start and goal as shown in (B) in figure 3.17.

3.4.1 Emergent Behavior

Emergence is often invoked in an almost mystical sense regarding the capabilities of behavior-based systems. Emergent behavior implies a holistic capability where the sum is considerably greater than its parts. It is true that what occurs in a behavior-based system is often a surprise to the system's designer, but does the surprise come because of a shortcoming of the analysis of the constituent behavioral building blocks and their coordination, or because of something else?

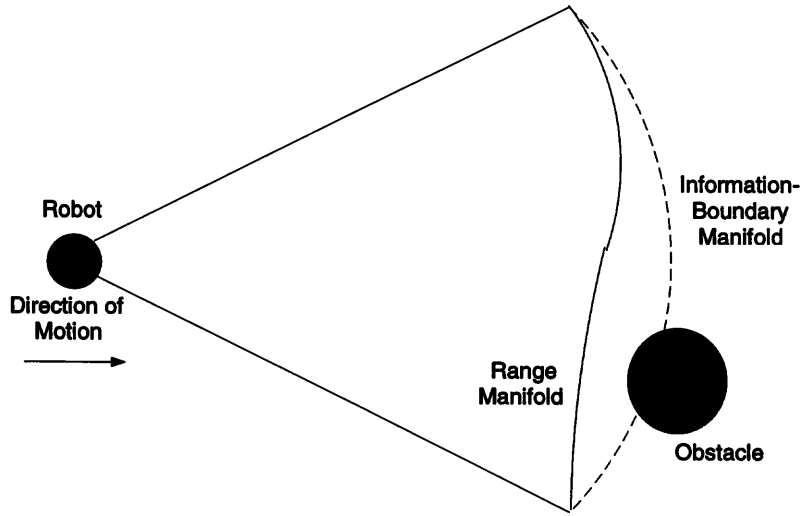


Figure 3.19
The use of manifolds for reactive collision avoidance.

The notion of emergence as a mystical phenomenon needs to be dispelled, but the concept in a well-defined sense can still be useful. Numerous researchers have discussed emergence in its various aspects:

- Emergence is "the appearance of novel properties in whole systems" (Moravec 1988).
- "Global functionality emerges from the parallel interaction of local behaviors" (Steels 1990).
- "Intelligence emerges from the interaction of the components of the system" (where the system's functionality, i.e., planning, perception, mobility, etc., results from the behavior-generating components) (Brooks 1991a).
- "Emergent functionality arises by virtue of interaction between components not themselves designed with the particular function in mind." (McFarland and Bosser 1993).

The common thread through all these statements is that emergence is a property of a collection of interacting components, in our case behaviors. The question arises however: Individual behaviors are well defined functionally, so why should a coordinated collection of them produce novel or unanticipated results?

Coordination functions as defined in this chapter are algorithms and hence contain no surprises and possess no magical perspective. In some cases, they are straightforward, such as choosing the highest ranked or most dominant behavior; in others they may be more complex, involving fusion of multiple active behaviors. Nonetheless, they are generally deterministic functions and certainly computable. Why then does the ineffable quality of emergence arise when these behavior-based robotic systems are released in the world? Why can we not predict their behavior exactly?

The answer to this question lies not within the robot itself but rather in the relationship the robot has with its environment. For most situations in which the behavior-based paradigm is applied, the world itself resists analytical modeling. Nondeterminism is rampant. The real world is filled with uncertainty and dynamic properties. Further, the perception process itself is also poorly characterized: Precise sensor models for open worlds do not exist. If a world model could be created that accurately captured *all* of its properties then emergence would not exist: Accurate predictions could be made. But it is the nature of the world to resist just such characterization, hence we cannot predict a priori, with any degree of confidence, in all but the simplest of worlds, how the world will present itself. Probabilistic models can provide guidance but not certainty.

For example, simply consider all the things involved in something as simple as your attending a class. Lighting conditions and weather will affect perceptual processing, the traffic on the road and sidewalks can be characterized only weakly (i.e., you do not know ahead of time the location and speeds of all people and cars that you meet). The complexities of the world resist modeling, leading to the aphorism espoused by Brooks (1989b): "The world is its own best model." Since the world cannot be faithfully modeled and is full of surprises itself, it is small wonder that behavior-based systems, which are tightly grounded to the world, reflect these surprises to their designers and observers by performing unexpected or unanticipated actions (or by definition actions not explicitly captured in the designer's intentions).

Summarizing, emergent properties are a common phenomena in behavior-based systems, but there is nothing mystical about them. They are a consequence underlying the complexity of the world in which the robotic agent resides and the additional complexity of perceiving that world. Let us now move on to methods for expressing the coordination of behavior within these robotic systems.

3.4.2 Notation

We now consider the case where multiple behaviors may be concurrently active within a robotic system. Defining additional notation, let

- **S** denote a vector of all stimuli $\mathbf{s}_i$ relevant for each behavior β_i detectable at time t.
- **B** denote a vector of all active behaviors β_i at a given time t.
- **G** denote a vector encoding the relative strength or gain g_i of each active behavior β_i.
- **R** denote a vector of all responses $\mathbf{r}_i$ generated by the set of active behaviors.

A new behavioral coordination function, **C**, is now defined such that

$$\rho = \mathbf{C}(\mathbf{G} * \mathbf{B}(\mathbf{S})),$$

or alternatively

$$\rho = \mathbf{C}(\mathbf{G} * \mathbf{R})$$

where

$$\mathbf{R} = \begin{bmatrix} \mathbf{r}_1 \\ \mathbf{r}_2 \\ \vdots \\ \mathbf{r}_n \end{bmatrix} \quad \mathbf{S} = \begin{bmatrix} \mathbf{s}_1 \\ \mathbf{s}_2 \\ \vdots \\ \mathbf{s}_n \end{bmatrix} \quad \mathbf{G} = \begin{bmatrix} g_1 \\ g_2 \\ \vdots \\ g_n \end{bmatrix}, \text{ and } \mathbf{B} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_n \end{bmatrix},$$

and where $*$ denotes the special scaling operation for multiplication of each scalar component (g_i) by the corresponding magnitude of the component vector ($\mathbf{r}_i$), resulting in a column vector $\mathbf{r}'_i$ of the same dimension as $\mathbf{r}_i$. In other words, $\mathbf{r}'_i$ represents a response reaction in the same direction as $\mathbf{r}_i$ with its magnitude scaled by g_i. (Alternatively, **G** can be represented as a diagonal matrix).

Restating, the coordination function **C**, operating over all active behaviors **B**, modulated by the relative strengths of each behavior specified by the gain vector **G**, for a given set of detected stimuli at time t, **S**, produces the overall robotic response ρ, where ρ is the vector encoding the global response that the robot will undertake, represented in the same form as r (e.g., $[\mathbf{x}, \mathbf{y}, \mathbf{z}, \theta, \phi, \psi]$).

C can be arbitrarily defined, but several strategies are commonly used to encode this function. They are split across two dimensions: competitive and cooperative. The simplest competitive method is pure arbitration, where only one behavior's output ($\mathbf{r}_i$) is selected from **R** and assigned to ρ, that is, arbitrarily choosing only one response from the many available. Several methods have

been used to implement this particular technique, including behavioral prioritization (subsumption) or action selection. Cooperative methods, on the other hand, blend the outputs of multiple behaviors in some way consistent with the agent's overall goals. The most common method of this type is vector addition. Competitive and cooperative methods can be composed as well. Section 3.4.3 examines these methods in more detail.

First, however, let us revisit the classroom example. Given the robot's current perceptions at time t:

$$\mathbf{S} = \begin{bmatrix} (class - location, 1.0) \\ (detected - object, 0.2) \\ (detected - student, 0.8) \\ (detected - path, 1.0) \\ (detected - elder, 0.0) \end{bmatrix},$$

for each $\mathbf{s}_i = (p, \lambda)$, where λ is the stimulus p's percentage of maximum strength. The situation above indicates that the robot knows exactly where the classroom is, has detected an object a good distance away, sees a student approaching nearby, sees the sidewalk (path) with certainty, and senses that there are no elders nearby.

We can then represent the behavioral response as:

$$\mathbf{B}(\mathbf{S}) = \begin{bmatrix} \beta_{move-to-class}(\mathbf{s}_1) \\ \beta_{avoid-object}(\mathbf{s}_2) \\ \beta_{dodge-student}(\mathbf{s}_3) \\ \beta_{stay-right}(\mathbf{s}_4) \\ \beta_{defer-to-elder}(\mathbf{s}_5) \end{bmatrix}.$$

$\mathbf{R}$ then is computed using each β,

$$\mathbf{R} = \begin{bmatrix} \mathbf{r}_{move-to-class} \\ \mathbf{r}_{avoid-object} \\ \mathbf{r}_{dodge-student} \\ \mathbf{r}_{stay-right} \\ \mathbf{r}_{defer-to-elder} \end{bmatrix}$$

with component vector magnitudes equal to (arbitrarily for this case)

$$\mathbf{R}_{magnitude} = \begin{bmatrix} 1.0 \\ 0 \\ 0.8 \\ 1.0 \\ 0 \end{bmatrix}$$

where each $\mathbf{r}_i$ encodes an $[x, y, \theta]$ for this particular robot expressing the desired directional response for each independent behavior. In the example, *avoid-object* and *defer-to-elder* are below threshold and generate no response, whereas *move-to-class* and *stay-right* are at maximum strength (defined as 1.0 here). Remember that the response is computed from β: $S \rightarrow R$.

Before the coordination function **C** is applied, **R** is multiplied by the gain vector **G**. For the numbers used in the example for **G**, *stay-right* is least important ($g_{stay-right} = 0.4$), *dodge-student* is deemed the most important behavior ($g_{dodge-student} = 1.5$), while *move-to-class* and *defer-to-elder* are of equal intermediate priority ($g_{move-to-class} = g_{defer-to-elder} = 0.8$) and the *avoid-object* behavior ranks second overall ($g_{avoid-object} = 1.2$).

$$\mathbf{R}' = \mathbf{G} * \mathbf{R},$$

where:

$$\mathbf{G} = \begin{bmatrix} g_{move-to-class} \\ g_{avoid-object} \\ g_{dodge-student} \\ g_{stay-right} \\ g_{defer-elder} \end{bmatrix} = \begin{bmatrix} 0.8 \\ 1.2 \\ 1.5 \\ 0.4 \\ 0.8 \end{bmatrix}$$

yielding:

$$\rho = \mathbf{C}(\mathbf{R}') = \mathbf{C}\left(\begin{bmatrix} g_1 * \mathbf{r}_1 \\ g_2 * \mathbf{r}_2 \\ g_3 * \mathbf{r}_3 \\ g_4 * \mathbf{r}_4 \\ g_5 * \mathbf{r}_5 \end{bmatrix}\right)$$

with scaled component vector magnitudes

$$\mathbf{R}'_{magnitude} = \begin{bmatrix} 0.8 \\ 0 \\ 1.2 \\ 0.4 \\ 0 \end{bmatrix}$$

G is of value only when multiple behaviors are being coordinated, as it enables the robot to set priorities through establishing the relative importance of each of its constituent behaviors.

If a simple winner-take-all coordination strategy is in place (action-selection), a single component vector $g_i\mathbf{r}_i$ (based on some metric such as

greatest magnitude) is chosen by **C**, assigned to ρ and executed. In the example above, this is the component with a value of 1.2, associated with the *dodge-student* behavior. Thus the response undertaken using this simple arbitration function is the action required to dodge an oncoming student. Note that this behavior dominates only given the current perceptual readings at time t and that the behavioral response will change as the robot's perceptions of the world also change. For a priority-based arbitration system, the highest-ranked component behavior encoded in **G** (above threshold) would be chosen and executed, independent of the individual components' relative magnitudes. In our example, *dodge-student* would also be chosen using this method, assuming the stimulus is above threshold, since $g_{dodge-student}(1.5)$ is the highest-ranked behavior.

Finally, coordination functions are recursively defined to operate not only on low-level behavioral responses but also on the output of other coordination operators (which also are behavioral responses).

$$\rho' = \mathbf{C}'\left(\begin{bmatrix} \rho_1 \\ \rho_2 \\ \vdots \\ \rho_n \end{bmatrix}\right),$$

where ρ' coordinates the output of lower-level coordinative functions. One benefit of this definition is the ability to sequence sets of behavioral activities for a robot temporally.

3.4.3 Behavioral Coordination

We now turn to examine the nature of **C**, the coordination function for behaviors. This function has two predominant classes, competitive and cooperative, each of which has several different strategies for realization. The different classes may be composed together, although frequently in the behavior-based architectures described in chapter 4, a commitment is made to one type of coordination function specific to a particular approach.

3.4.3.1 Competitive Methods

Conflict can result when two or more behaviors are active, each with its own independent response. Competitive methods provide a means of coordinating behavioral response for conflict resolution. The coordinator can often be viewed

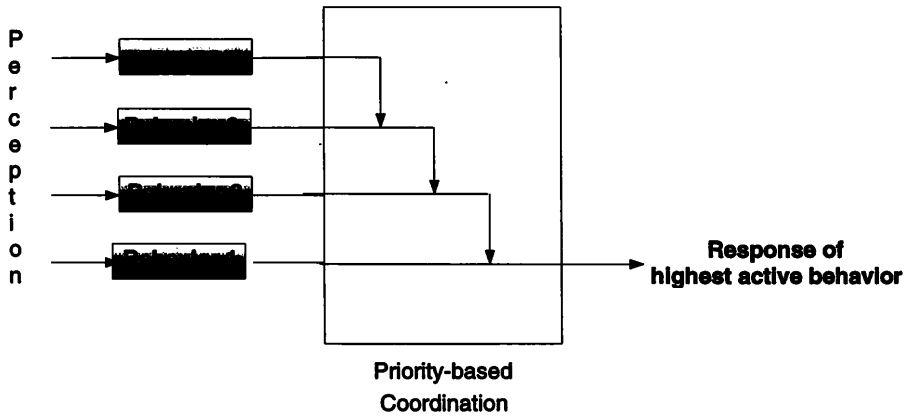


Figure 3.20
Arbitration via suppression network.

as a winner-take-all network in which the single response for the winning behavior out-muscles all the others and is directed to the robot for execution. This type of competitive strategy can be enacted in a variety of ways.

Arbitration requires that a coordination function serving as an arbiter select a single behavioral response. The arbitration function can take the form of a fixed prioritization network in which a strict behavioral dominance hierarchy exists, typically through the use of suppression and inhibition in a descending manner. This is a hallmark of subsumption-based methods (Brooks 1986), the particulars of which are described in chapter 4. Figure 3.20 shows an example illustrated as a dominance hierarchy.

Action-selection methods (Maes 1990) arbitrarily select the output of a single behavior, but this is done in a less autocratic manner. Here the behaviors actively compete with each other through the use of activation levels driven by both the agent's goals (or intentionality) and incoming sensory information. No fixed hierarchy is established; rather the behavioral processes compete with each other for control at a given time. Run time arbitration occurs by the selection of the most active behavior, but no predefined hierarchy is present. Figure 3.21 captures this notionally. The concept of lateral inhibition (section 2.2.1) can easily be implemented using this method where one behavior's strong output negatively affects another's output.

Another even more democratic competitive method involves behaviors' generating votes for actions, with the action that receives the most votes being the single behavior chosen (Rosenblatt and Payton 1989). This technique

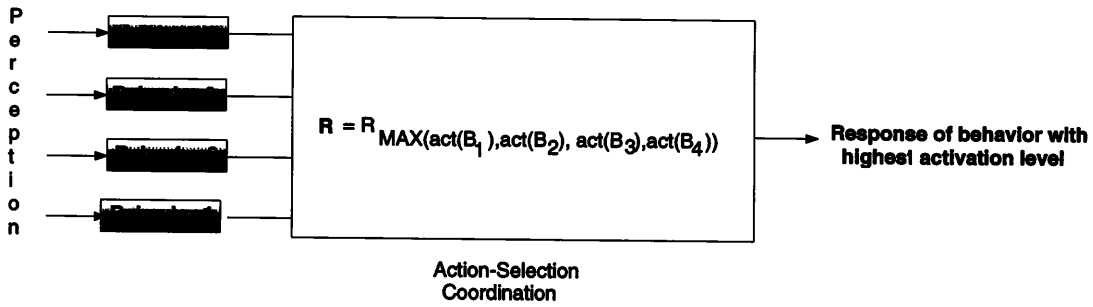


Figure 3.21
Arbitration via action-selection.

is embodied in the Distributed Architecture for Mobile Navigation (DAMN) (Rosenblatt 1995). Here instead of each behavior's being encoded as a set of rule-based responses, each behavior casts a number of votes toward a predefined set of discrete motor responses. For navigation of an unmanned ground vehicle, the behavioral response set for steering consists of

{*hard-left*, *soft-left*, *straight-ahead*, *soft-right*, *hard-right*}.

Each active behavior (e.g., *goal-seeking, obstacle-avoidance, road-following, cross-country, teleoperation, map-based navigation*) has a certain number of votes (g_i) and a user-provided distribution for allocating those votes. Arbitration takes place through a winner-take-all strategy in which the single response with the most votes is enacted. In a sense, this allows a level of behavioral cooperation, but the method is still arbitrary in its choice of a single response. Figure 3.22 depicts this type of arbitration method.

3.4.3.2 Cooperative Methods

Cooperative methods provide an alternative to competitive methods such as arbitration. Behavioral fusion provides the ability to use concurrently the output of more than one behavior at a time.

The central issue in combining the outputs of behaviors is finding a representation amenable to fusion. The potential-fields method, as described in section 3.3.2, provides one useful formalism. As shown earlier, the most straightforward method is through vector addition or superpositioning. Each behavior's relative strength or gain, g_i, is used as a multiplier of the vectors before addition. Figure 3.23 illustrates how this is accomplished.

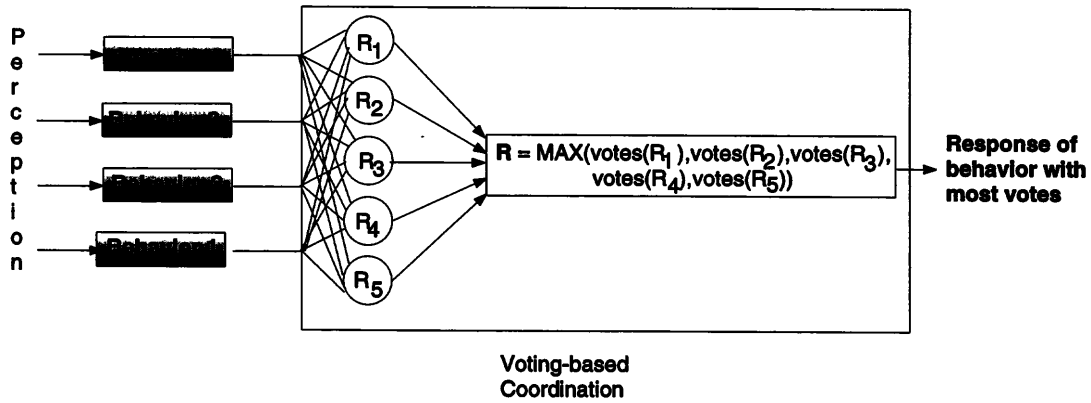


Figure 3.22
Arbitration via voting.

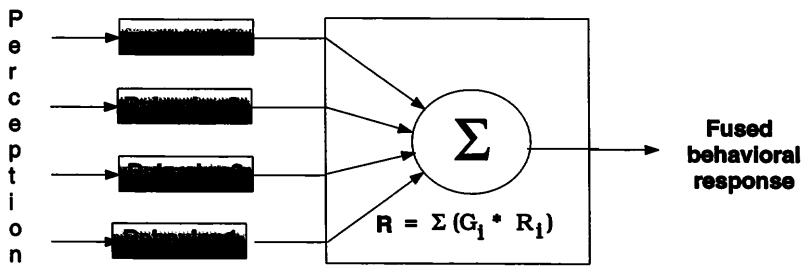


Figure 3.23
Behavioral fusion via vector summation.

Figure 3.24 shows a field with three active obstacles and a goal located in the lower right corner. In (A), the goal attractor behavior has twice the obstacle avoidance behavior's relative strength. The path the robot takes through this field cuts fairly close to the obstacles themselves, resulting in a shorter although more perilous path. In (B), the obstacle avoidance has twice the strength of the goal attraction. The path taken is considerably longer but also further from the obstacles themselves. Vector addition provides a continuum for combining these fields controllable through the gain vector **G**.

In work at Hughes Artificial Intelligence Center (Payton et al. 1992), the issue of combining multiple disparate behavioral outputs is handled by avoiding

the sole use of single discrete response values. Instead, the responses of each behavior constrain the control variables as follows:

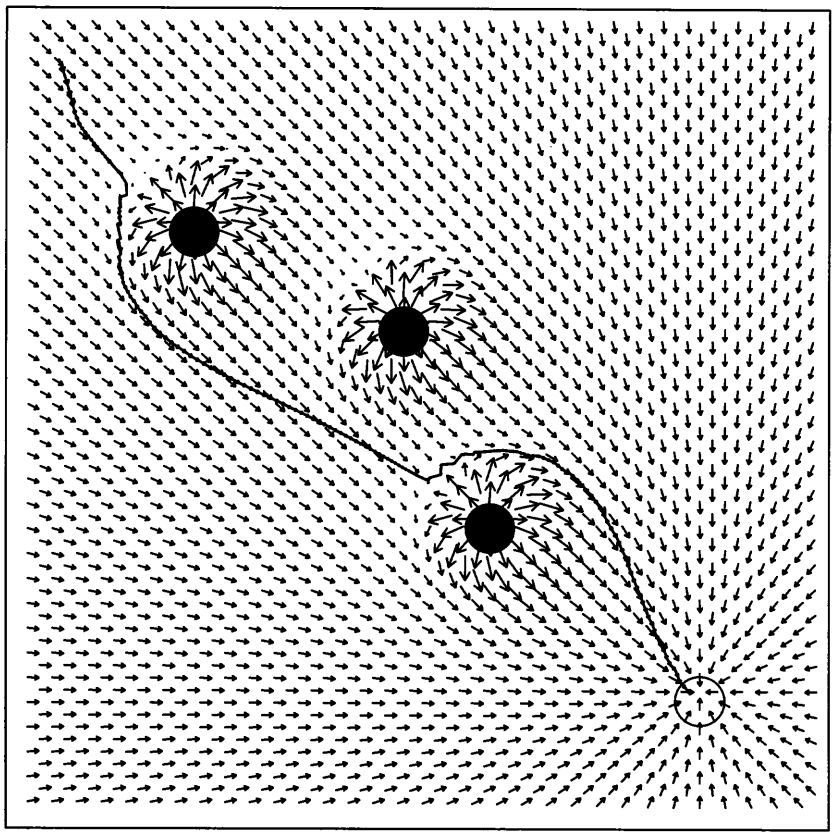
- Zone: establishment of upper and lower bounds for a control variable
- Spike: single value designation resulting in standard priority-based arbitration
- Clamp: establishment of upper or lower bound on a control variable

In essence, the system provides constraints within which the control system must operate to satisfy its behavioral requirements. The algorithm for fusing the behaviors is as follows:

```
Generate a profile by summing each active behavior's zones and
spikes
If maximum is a spike
     choose that command
else (maximum is in zone)
     choose command requiring minimal change from current
     control setting
If command chosen beyond a clamped value
     choose most dominant clamp from all clamps
     If that clamp dominates profile
          move command value to just within clamped region
```

In this manner, commands can be described in the control variables for each actuator (e.g., speed of a motor, angles of a control surface, etc). Clamps limit the acceptable ranges for each actuator. This particular system has been applied to the control of an underwater robot using high-level behaviors such as stealth, safety, urgency, and efficiency, which are mapped onto low-level behaviors that control the vehicle's speed, the angle of the dive plane, proximity to the ocean bottom, and so forth, which in turn control the vehicle actuators, such as the ballast pumps and dive plane motor.

Formal methods for expressing behavioral blending are discussed in Saffiotti, Konolige, and Ruspini 1995. In this approach, each behavior is assigned a desirability function that can be combined using specialized logical operators called t-norms, a form of multivalued logic. A single action is chosen from a set of actions created by blending the weighted primitive action behaviors. As this is essentially a variation of fuzzy control, where blending is the fuzzification operation and selection is the defuzzification process, the method for generating these results is deferred until chapter 8.



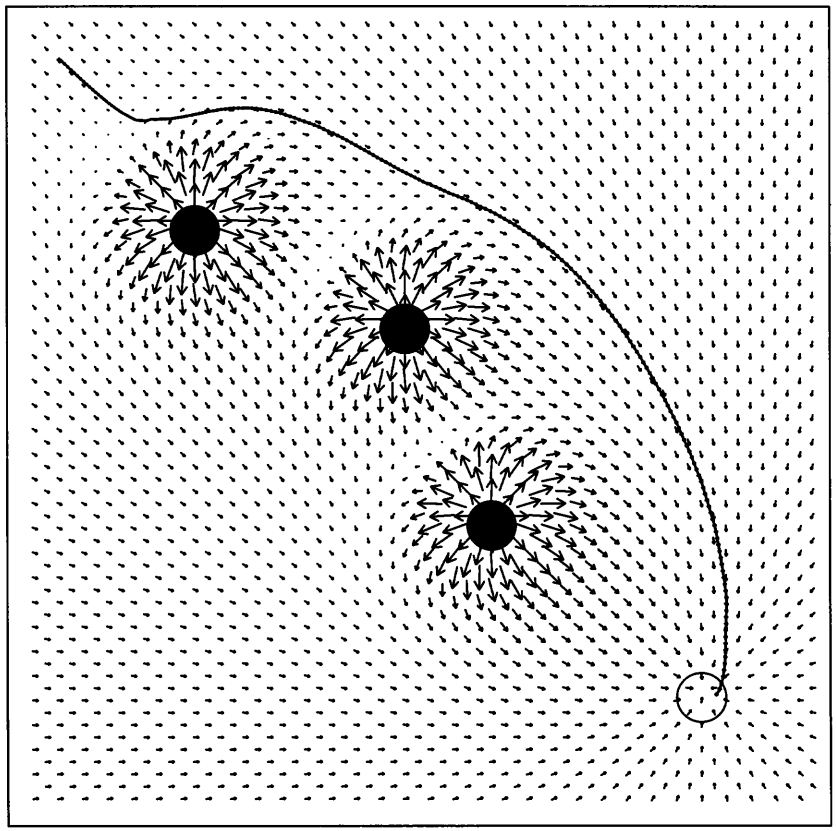
(A)

Figure 3.24

It is also possible that the deformation manifolds approach described for obstacle avoidance in section 3.3.2 could be extended to behavioral fusion as the approach potentially can provide a common representation for robotic action. So far, however, it has been limited to arbitration.

3.4.4 Behavioral Assemblages

Behavioral assemblages are the packages from which behavior-based robotic systems are constructed. An assemblage is recursively defined as a coordinated collection of primitive behaviors or assemblages. Each individual assemblage consists of a coordination operator and any number of behavioral components



(B)

Figure 3.24 ***(continued)***
Behavioral fusion: (A) goal attraction dominates; (B) obstacle avoidance dominates.

(primitives or assemblages). The power of assemblage use arises from the notion of abstraction, in which we can create higher-level behaviors from simpler ones and refer to them henceforward independent of their constituent elements. Abstraction has traditionally been a powerful concept in AI (Sacerdoti 1974), and it is no less so here. Abstraction enables us to reuse assemblages in an easy, modular manner to construct behavior-based systems; provides the ability to reason over them for use in hybrid architectures (chapter 6); and provides coarser levels of granularity for adaptation and learning methods to be applied (chapter 8).

We have previously used the notation $\mathbf{C(G * B(S))} = \rho$, in section 3.4.2. We denote an assemblage q_i such that q_i symbolizes $\mathbf{C(G * B(S))}$, hence $q_i = \rho$. For convenience, we will generally omit the ρ in assemblage expression, leaving q_i, which maps conveniently onto a state within an FSA diagram. Although assemblages can be expressed in any of the notational formats section 3.4.2 describes, whenever there is a temporal component (i.e., the behavioral structure of the system changes over time), FSA notation is used most often.

Figure 3.12 has already shown an assemblage of assemblages in FSA form for a competition robot. The constituent behaviors for two of that robot's assemblage states are

- move-to-pole
- move-to-goal(detect-pole)
- avoid-static-obstacle(detect-obstacles)
- noise(generate-direction) low gain
- wander
- probe(detect-open-area)
- avoid-static-obstacle(detect-obstacles)
- noise(generate-direction) high gain
- avoid-past(detect-visited-areas)

Each of these FSA states can be equivalently depicted as an SR diagram, an example of which appears in figure 3.25.

All of these assemblages can, in turn, be bundled together in a high-level SR diagram if desired (figure 3.26). Using this alternate representation, which has merely a sequencer label for the coordination function, the explicit temporal dependencies between states (i.e., the state transition function q associated with the perceptual triggers) are lost when compared to the earlier FSA version (figure 3.12).

Another form of assemblage construction, referred to rather as "hierarchically mediated behaviors," appears in Kaelbling 1986. A similar hierarchical abstraction capability also appears in teleo-reactive systems (Nilsson 1994). In RS (Lyons and Arbib 1989), an assemblage is defined as a network of schemas that can be viewed as a schema itself. In all these cases, behavioral aggregations (or assemblages) can be viewed recursively (or alternatively, hierarchically) as behaviors themselves.

Assemblages, defined as hierarchical recursive behavioral abstractions, constitute the primary building blocks of behavior-based robotic systems and are ultimately grounded in primitive behaviors attached to sensors and actuators.

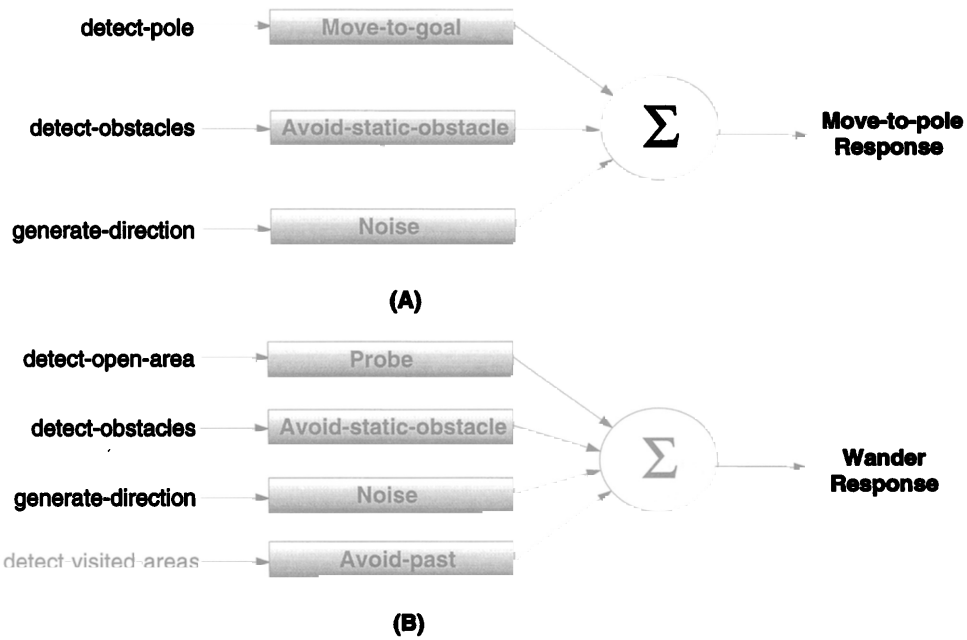


Figure 3.25
SR diagram for the move-to-pole and wander assemblages.

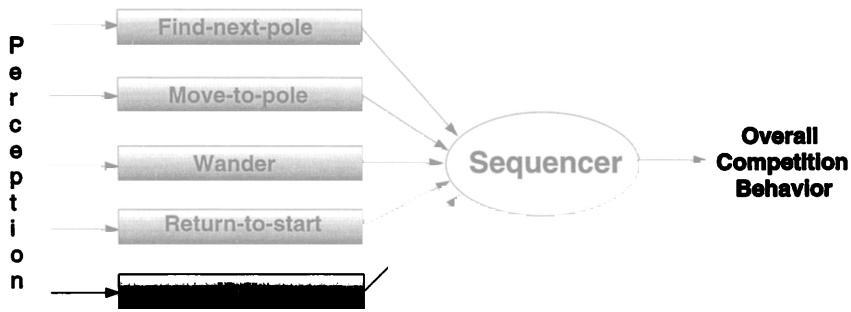


Figure 3.26
Competition SR diagram.

3.5 CHAPTER SUMMARY

- Robotic behaviors generate a motor response from a given perceptual stimulus.
- Purely reactive systems avoid the use of explicit representational knowledge.
- Behavior-based systems are inherently modular in design and provide the ability for software reuse.
- Biological models often serve as the basis for the design of behavior-based robotic systems.
- Three design paradigms for building behavior-based systems have been presented: ethologically guided/constrained design, using biological models as the basis for behavioral selection, design and validation; situated activity, which creates behaviors that fit specific situational contexts in which the robot will need to respond; and experimentally driven design, which uses a bottom-up design strategy based on the need for additional competency as the system is being built.
- The expression of behaviors can be accomplished in several different ways: SR diagrams, which intuitively convey the flow of control within a behavior-based system; functional notation, which is amenable to the generation of code for implementation; and FSA diagrams, which are particularly well suited for representing behavioral assemblages' time-varying composition.
- Other, more formal methods have been developed for expressing behaviors, such as *RS* and situated automata as epitomized by the Gapps language.
- Behaviors can be represented as triples (S, R, β), with S being the stimulus domain, R the range of response, and β the behavioral mapping between them.
- The presence of a stimulus is necessary but not sufficient to evoke a motor response in a behavior-based robot. Only when the stimulus exceeds some threshold value τ will it produce a response.
- g_i, a strength multiplier or gain value, can be used to turn off behaviors or increase the response's relative strength.
- Responses are encoded in two forms: discrete encoding, in which an enumerable set of responses exists; or continuous functional encoding, in which an infinite space of responses is possible for a behavior.
  - Rule-based methods are often used for discrete encoding strategies.
  - Approaches based on the potential-fields method are often used for the continuous functional encoding of robotic response.
- There is nothing magical about emergent behavior; it is a product of the complexity of the relationship between a robotic agent and the real world that resists analytical modeling.

- Notational methods for describing assemblages and coordination functions used throughout the text have been presented.
- The two primary mechanisms for behavioral coordination are competitive or cooperative, but they can be combined if desired.
  - Competitive methods result in the selection of the output of a single behavior, typically either by arbitration or action-selection.
  - Cooperative methods often use superpositioning of forces or gradients generated from field–based methods, including potential field-based approaches or navigational templates.
- Assemblages are recursively defined aggregations of behaviors or other assemblages.
- Assemblages serve as important abstractions useful for constructing behavior-based robots.

Chapter 4
Behavior-Based Architectures

One can expect the human race to continue attempting systems just within or just beyond our reach; and software systems are perhaps the most intricate and complex of man's handiworks. The management of this complex craft will demand our best use of new languages and systems, our best adaptation of proven engineering management methods, liberal doses of common sense, and a God-given humility to recognize our fallibility and limitations.

–Frederick P. Brooks, Jr.

There are two ways of constructing a software design. One way is to make it so simple that there are obviously no deficiencies. And the other way is to make it so complicated that there are no obvious deficiencies.

—C.A.R. Hoare

Chapter Objectives

1. To characterize what a robot architecture is.
2. To understand the requirements for the design of a behavior-based robotic architecture.
3. To understand, in depth, two different reactive robotic architectures: subsumption and motor schema.
4. To review many of the other behavior-based architectural choices available to the robot system builder.
5. To develop design principles for the construction of a behavior-based robotic architecture.

4.1 WHAT IS A ROBOTIC ARCHITECTURE?

In chapter 3, we learned about robotic behaviors, including methods for expressing, encoding, and coordinating them. To design and build behavior-based robotic systems, commitments need to be made to the actual, specific methods to be used during this process. This need leads us to the study of robotic architectures: software systems and specifications that provide languages and tools for the construction of behavior-based systems.

All of the architectures described in this chapter are concerned with behavioral control. As described in chapter 1, several non–behavior-based robotic architectures appeared before the advent of reactive control, for example, NASREM (section 1.3.1). Here, however, we focus on behavior-based systems. Though considerably varied, these architectures share many common features:

- emphasis on the importance of coupling sensing and action tightly
- avoidance of representational symbolic knowledge
- decomposition into contextually meaningful units (behaviors or situation-action pairs)

Although these architectures share a common philosophy on the surface, there are many deep distinctions between them, including

- the granularity of behavioral decomposition
- the basis for behavior specification (ethological, situated activity, or experimental)
- the response encoding method (e.g., discrete or continuous)
- the coordination methods used (e.g., competitive versus cooperative)
- the programming methods, language support available, and the extent of software reusability.

In this chapter we study several common robotic architectures used to build behavior-based systems. Tables appear throughout summarizing the characteristics for each of the behavior-based architectures discussed. Two of the architectures have been singled out for closer scrutiny than the others: the subsumption architecture using rule-based encodings and priority-based arbitration; and motor schemas using continuous encoding and cooperative combination of vectors.

4.1.1 Definitions

Perhaps a good place to begin searching for our definition of robotic architectures would be with the definition of computer architectures. Stone (1980, p. 3), one of the best-known computer architects, uses the following definition: "Computer architecture is the discipline devoted to the design of highly specific and individual computers from a collection of common building blocks."

Robotic architectures are essentially the same. In our robotic control context, however, architecture usually refers to a software architecture, rather than the hardware side of the system. So if we modify Stone's definition accordingly, we get:

Robotic architecture is the discipline devoted to the design of highly specific and individual robots from a collection of common software building blocks.

How does this definition coincide with other working definitions by practicing robotic architects? According to Hayes-Roth (1995, p. 330), an architecture refers to " . . . the abstract design of a class of agents: the set of structural components in which perception, reasoning, and action occur; the specific functionality and interface of each component, and the interconnection topology between components." Although her discussion of agent architectures is targeted for artificially intelligent systems in general, it also holds for the subclass with which we are concerned, namely, behavior-based robotic systems. Indeed, a surveillance mobile robot system (figure 4.1) has been developed that embodies her architectural design principles (Hayes-Roth et al. 1995). She argues that architectures must be produced to fit specific operating environments, a concept closely related to our earlier discussion of ecological niches in chapter 2 and related to the claim in McFarland and Bosser 1993 that robots should be tailored to fit particular niches.

Mataric (1992a) provides another definition, stating, "An architecture provides a principled way of organizing a control system. However, in addition to providing structure, it imposes constraints on the way the control problem can be solved." One final straightforward definition is from Dean and Wellman (1991, p. 462): "An architecture describes a set of architectural components and how they interact."

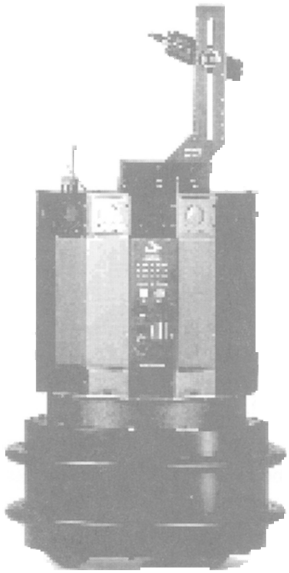


Figure 4.1
Nomad 200 robot, of the type used for surveillance at Stanford. (Photograph courtesy of Nomadic Technologies Inc., Mountain View, California.)

4.1.2 Computability

Existing robotic architectures are diverse, from the hierarchical NASREM architecture to purely reactive systems such as subsumption (section 4.3) to hybrid architectures (chapter 6). In what ways can instances chosen from the diversity of architectural solutions be said to differ from one another? In what ways can they be said to be the same?

The answer to these questions is related to the distinction between computability and organizing principles. Architectures are constructed from components, with each specific architecture having its own peculiar set of building blocks. The ways in which these building blocks can be connected facilitate certain types of robotic design in given circumstances. Organizing principles underlie a particular architecture's commitment to its component structure, granularity, and connectivity.

From a computational perspective, however, we may see that the various architectures are all equivalent in their computational expressiveness. Consider, for instance, the differences between programming languages. Different

choices are available to the programmer ranging from machine language to assembler to various high-level languages (such as Fortran, Cobol, C, Pascal, and LISP) to very high–level languages such as those used in visual programming. Is there any fundamental incompatibility in the idea that one language can do something that another cannot?

Consider the results that Bohm and Jacopini (1966) derived concerning computability in programming languages. They proved that if any language contains the three basic constructs of sequencing, conditional branching, and iteration, it can compute the entire class of computable functions (i.e., it is Turing equivalent). This essentially states that from a computational perspective the common programming languages have no differences.

The logical extension is that since all robotic architectures provide the capability to perform tasks sequentially, allow conditional branching, and provide the ability for iterative constructs, these architectures are computationally equivalent. All behavior-based robotic architectures are essentially software languages or frameworks for specifying and controlling robots. The level of abstraction they offer may differ, but not the computability.

This does not mean, of course that we will start writing AI programs in Cobol. It does mean that each current programming language has in turn found a niche in which it serves well and thus has survived (i.e., remained in usage) because it is well suited for that particular task. Some argue that the same holds for robotic architectures as well: each serves a particular domain (or niche) and will be subjected to the same environmental stresses for survival as are computer architectures or software languages.

Behavior-based robotic systems serve best when the real world cannot be accurately characterized or modeled. Whenever engineering can remove uncertainty from the environment, purely behavior-based systems may not necessarily afford the best solution for the task involved and hierarchical architectures (chapter 1) may prove more suitable, as, for example, in factory floor operations where the environment can be altered to fit the robot's needs. More often than not, however, much as we try, we cannot remove uncertainty, unpredictability, and noise from the world. Behavior-based robotic architectures were developed in response to this difficulty and choose instead to deal with these issues from the onset, relying heavily on sensing without constructing potentially erroneous global world models. The more the world changes during execution, the more the resulting value of any plan generated a priori decreases, and the more unstable any representational knowledge stored ahead of time or gathered during execution and remembered becomes.

At a finer level, we will see that behavior-based architectures, because of their different means of expressing behaviors and the sets of coordination functions they afford, provide significant diversity to a robotic system's designer. Each approach has its own strengths and weaknesses in terms of what it is best at doing or where it is most appropriately applied. The remainder of this chapter discusses a variety of behavior-based robot architectural solutions. Not all are expected to withstand the test of time, and many will likely suffer a fate similar to that of early programming languages (e.g., ALGOL, SNOBOL) and fade off into obscurity. Ecological pressure from sources ranging from ease of use for the designer to generalizability to public opinion to exogeneous factors (political, economic, etc.) will ultimately serve as the fundamental selection mechanism, not merely an academic's perspective on their elegance, simplicity, or utility.

As an aside, we note the recent controversy Penrose (1989, 1994) stirred up by claiming that no computer program can ever exhibit intelligence as according to Penrose, intelligence must incorporate solutions to noncomputable problems as well as those that are computable. Interestingly, he does not dismiss the attainment of intelligence as utterly impossible in a device and presents a novel, but rather speculative, approach based on quantum mechanics (a microtubule architecture if you will), rather than a computational approach, to achieve this goal. To say the least, his position has been strongly rebutted by many within the AI community and is often cursorily dismissed as rubbish. In the book's final chapter, we will revisit this issue of what intelligence means within the context of a robotic system and what we can or should expect from these systems. Suffice it to say, for now, that all the behavior-based architectures considered in this book are computational.

4.1.3 Evaluation Criteria

How can we measure an architecture's utility for a particular problem? A list of desiderata for behavior-based architectures is compiled below.

- **Support for parallelism**: Behavior-based systems are inherently parallel in nature. What kind of support does the architecture provide for this capability?
- **Hardware targetability**: Hardware targetability really refers to two different things. The first regards how well an architecture can be mapped onto real robotic systems, that is, physical sensors and actuators. The second is concerned with the computational processing. Chip-level hardware implementations are often preferred over software from a performance perspective. What

type of support is available to realize the architectural design in silicon (e.g., compilers for programmable logic arrays [Brooks 1987b])?

- **Niche targetabilty**: How well can the robot be tailored to fit its operating environment (Hayes-Roth 1995)? How can the relationships between robot and environment be expressed to ensure successful niche occupation?
- **Support for modularity**: What methods does an architecture provide for encapsulating behavioral abstractions? Modularity can be found at a variety of levels. By providing abstractions for use over a wide range of behavioral levels (primitives, assemblages, agents), an architecture makes a developer's task easier and facilates software reuse (Mataric 1992a).
- **Robustness**: A strength of behavior-based systems is their ability to perform in the face of failing components (e.g., sensors, actuators, etc.) (Payton et al. 1992; Horswill 1993a; Ferrell 1994). What types of mechanisms does the architecture provide for such fault tolerance?
- **Timeliness in development**: What types of tools and development environments are available to work within the architectural framework? Is the architecture more of a philosophical approach, or does it provide specific tools and methods for generating real robotic systems?
- **Run time flexibility**: How can the control system be adjusted or reconfigured during execution? How easily is adaptation and learning introduced?
- **Performance effectiveness**: How well does the constructed robot perform its intended task(s)? This aspect also encompasses the notion of timeliness of execution, or how well the system can meet established real-time deadlines. In other instances, specific quantitative metrics can be applied for evaluation purposes within a specific task context (Balch and Arkin 1994). These may include such things as time to task completion, energy consumption, minimum travel, and so forth, or combinations thereof.

These widely ranging criteria can be used for evaluating the relative merits of many of the architectures described in the remainder of this chapter.

4.1.4 Organizing Principles

From the discussion in chapter 3, several different dimensions for distinguishing robotic architectures become apparent, including

- Different coordination strategies, of particular note, competitive (e.g., arbitration, action-selection, voting) versus cooperative (e.g., superpositioning)
- Granularity of behavior: microbehaviors such as those found in situated activity-based systems (e.g., Pengi) or more general purpose task descriptions (e.g., RAPs).

- Encoding of behavioral response: discrete, that is, a prespecified set of possible responses (e.g., rule-based systems or DAMN), or continuous (e.g., potential field–based methods).

The remainder of this chapter first discusses two architectures in some detail: the subsumption architecture and motor schema–based systems (the reactive component of the Autonomous Robot Architecture (AuRA). Next it reviews several other significant behavior-based architectures, although at a higher level. Finally it presents design principles for constructing a behavior-based robotic system, in an architecture-independent manner as much as possible.

4.2 A FORAGING EXAMPLE

To ground the following architectural discussions, let us consider a well-studied problem in robotic navigation: foraging. This task consists of a robot's moving away from a home base area looking for attractor objects. Typical applications might include looking for something lost or gathering items of value. Upon detecting the attractor, the robot moves toward it, picks it up and then returns it to the home base. It repeats this sequence of actions until it has returned all the attractors in the environment. This test domain has provided the basis for a wide range of results on both real robots (Balch et al. 1995; Mataric 1993a) and in simulation. Foraging also correlates well with ethological studies, especially in the case of ants (e.g., Goss et al. 1990).

Several high-level behavioral requirements to accomplish this task include:

1. *Wander*: move through the world in search of an attractor
2. *Acquire*: move toward the attractor when detected
3. *Retrieve:* return the attractor to the home base once acquired

Figure 4.2 represents these higher-level assemblages. Each assemblage shown is manifested with different primitive behaviors and coordinated in different ways as we move from one architectural example to the next.

4.3 SUBSUMPTION ARCHITECTURE

Rodney Brooks developed the subsumption architecture in the mid-1980s at the Massachusetts Institute of Technology. His approach, a purely reactive behavior-based method, flew in the face of traditional AI research at the time. Brooks argued that the *sense-plan-act* paradigm used in some of the first autonomous robots such as Shakey (Nilsson 1984) was in fact detrimental to

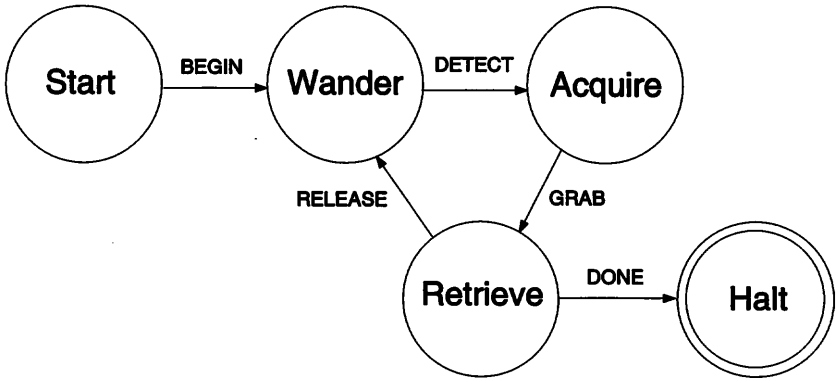


Figure 4.2
FSA diagram for foraging.

the construction of real working robots. He further argued that building world models and reasoning using explicit symbolic representational knowledge at best was an impediment to timely robotic response and at worst actually led robotics researchers in the wrong direction.

In his seminal paper, Brooks (1986) advocated the use of a layered control system, embodied by the subsumption architecture but layered along a different dimension than what traditional research was pursuing. Figure 4.3 shows the distinction, with the conventional *sense-plan-act* vertical model illustrated in (A) and the new horizontal decomposition in (B). (The orientation of the lines that separate the components determines vertical and horizontal.)

Much of the presentation and style of the subsumption approach is dogmatic. Tenets of this viewpoint include

- Complex behavior need not necessarily be the product of a complex control system.
- Intelligence is in the eye of the observer (Brooks 1991a).
- The world is its own best model (Brooks 1991a).
- Simplicity is a virtue.
- Robots should be cheap.
- Robustness in the presence of noisy or failing sensors is a design goal.
- Planning is just a way of avoiding figuring out what to do next (Brooks 1987a).
- All onboard computation is important.
- Systems should be built incrementally.
- No representation. No calibration. No complex computers. No high-bandwidth communication (Brooks 1989b).

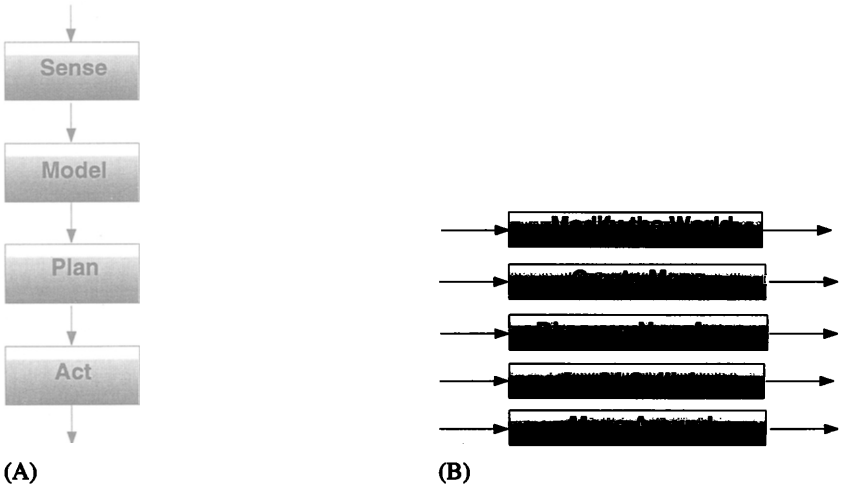


Figure 4.3
(A) Sense-plan-act model. (B) Subsumption (reactive) model.

This was hard to swallow for many in the AI community (and in many cases, still is). Brooks lobbied long and hard for rethinking the way intelligent robots in particular, and intelligent systems in general, should be constructed. This stance changed the direction of autonomous robotics research. Although currently many in the AI community take a more tempered position regarding the role of deliberation and symbolic reasoning (chapter 6), Brooks has not to date disavowed in print any of these principles (1991a).

Let us now move to the specifics of the subsumption architecture. Table 4.1 is the first of many tables throughout this chapter that provide a snapshot view of the design characteristics of a particular architecture in light of the material discussed in chapter 3.

4.3.1 Behaviors in Subsumption

Task-achieving behaviors in the subsumption architecture are represented as separate layers. Individual layers work on individual goals concurrently and asynchronously. At the lowest level, each behavior is represented using an augmented finite state machine (AFSM) model (figure 4.4). The AFSM encapsulates a particular behavioral transformation function β_i. Stimulus or response signals can be suppressed or inhibited by other active behaviors. A reset input is also used to return the behavior to its start conditions. Each AFSM performs

Table 4.1
Subsumption Architecture

Name	Subsumption architecture
Background	Well-known early reactive architecture
Precursors	Braitenberg 1984; Walter 1953; Ashby 1952
Principal design method	Experimental
Developer	Rodney Brooks (MIT)
Response encoding	Predominantly discrete (rule based)
Coordination method	Competitive (priority-based arbitration via inhibition and suppression)
Programming method	Old method uses AFSMs; new method uses Behavior Language
Robots fielded	Allen, Genghis (hexapod), Squirt (very small), Toto, Seymour, Polly (tour guide), several others
References	Brooks 1986; Brooks 1990b; Horswill 1993a

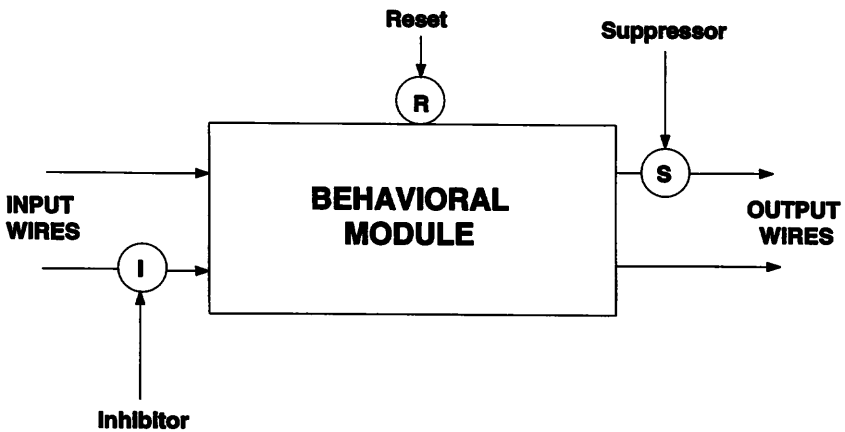


Figure 4.4
Original AFSM as used within the subsumption architecture.

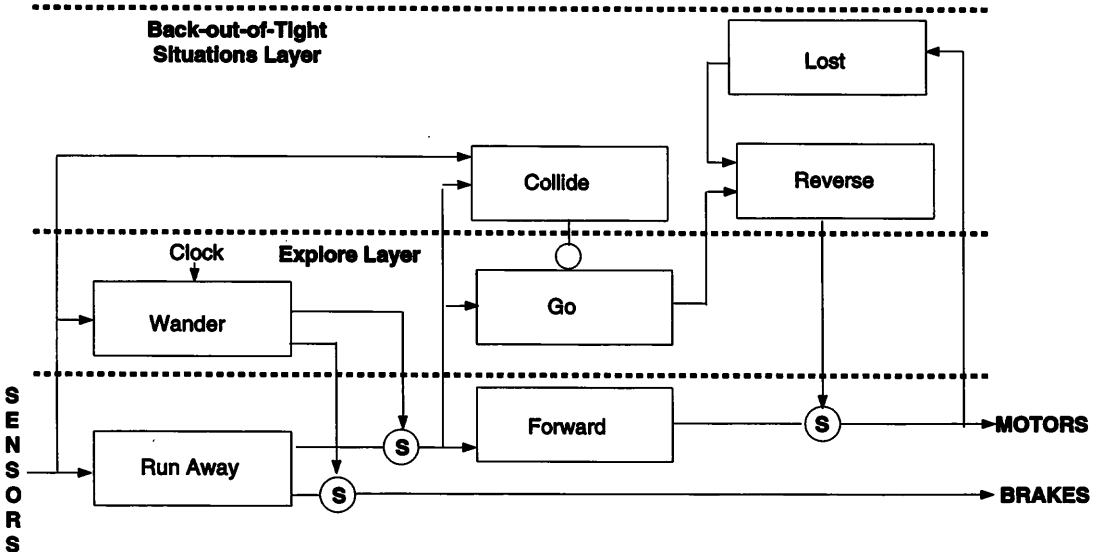


Figure 4.5
AFSMs for a simple three-layered robot (Brooks 1987b).

an action and is responsible for its own perception of the world. There is no global memory, bus, or clock. With this design, each behavioral layer can be mapped onto its own processor (Brooks 1987b). There are no central world models or global sensor representations. Required sensor inputs are channeled to the consuming behavior.

Figure 4.5 shows a simple robot with three behavioral layers. The system was implemented on a radio-controlled toy car (Brooks 1987b). The lowest behavior layer, *avoid-objects,* either halts or turns away from an obstacle, depending upon the input from the robot's infrared proximity sensors. The *explore* layer permits the robot to move in the absence of obstacles and cover large areas. The highest layer, *back-out-of-tight-situations*, enables the robot to reverse direction in particularly tight quarters where simpler avoidance and exploration behaviors fail to extricate the robot.

As can be seen, the initial subsumption language, requiring the specification of low-level AFSMs, was unwieldy for those not thoroughly schooled in its usage. Recognizing this problem, Brooks (1990a) developed the Behavior Language, which provides a new abstraction independent of the AFSMs themselves using a single rule set to encode each behavior. This high-level language is then compiled to the intermediate AFSM representation, which can then be further compiled to run on a range of target processors.

4.3.2 Coordination in Subsumption

The name subsumption arises from the coordination process used between the layered behaviors within the architecture. Complex actions subsume simpler behaviors. A priority hierarchy fixes the topology. The lower levels in the architecture have no awareness of higher levels. This provides the basis for incremental design. Higher-level competencies are added on top of an already working control system without any modification of those lower levels.

The older version of subsumption specified the behavioral layers as collections of AFSMs, whereas the newer version uses behavioral abstractions (in the form of rules) to encapsulate a robot's response to incoming sensor data. These abstractions are then compiled into the older AFSM form, but this step is transparent to the developer.

Coordination in subsumption has two primary mechanisms:

- Inhibition: used to prevent a signal being transmitted along an AFSM wire from reaching the actuators.
- Suppression: prevents the current signal from being transmitted and replaces that signal with the suppressing message.

Through these mechanisms, priority-based arbitration is enforced.

Subsumption permits communication between layers but restricts it heavily. The allowable mechanisms have the following characteristics:

- low baud rate, no handshaking
- message passing via machine registers
- output of lower layer accessible for reading by higher level
- inhibition prevents transmission
- suppression replaces message with suppressing message
- reset signal restores behavior to original state

The world itself serves as the primary medium of communication. Actions taken by one behavior result in changes within the world and the robot's relationship to it. New perceptions of those changes communicate those results to the other behaviors.

4.3.3 Design in Subsumption-Based Reactive Systems

The key aspects for design of subsumption-style robots are situatedness and embodiment (Brooks 1991b). Situatedness refers to the robot's ability to sense its current surroundings and avoid the use of abstract representations, and em-

bodiment insists that the robots be physical creatures and thus experience the world directly rather than through simulation. Mataric 1992a presents heuristics for the design and development of this type of robot for a specific task. The basic procedure outlined is as follows:

1. Qualitatively specify the behavior needed for the task, that is, describe the overall way the robot responds to the world.
2. Decompose and specify the robot's independent behaviors as a set of observable disjoint actions by decomposing the qualitative behavior specified in step 1.
3. Determine the behavioral granularity (i.e., bound the decomposition process) and ground the resulting low-level behaviors onto sensors and actuators.

An additional guideline regarding response encoding recommends the use of small motions rather than large ballistic ones by resorting to frequent sensing. Finally, coordination is imposed by initially establishing tentative priorities for the behaviors and then modifying and verifying them experimentally.

Let us now review the example of experimentally driven subsumption-style design (Brooks 1989a) previously mentioned in section 3.1.3. The target robot is a six-legged walking machine named Genghis (figure 3.6). Its high-level behavioral performance is to be capable of walking over rough terrain and to have the ability to follow a human. This constitutes the qualitative description of task-level performance mentioned in step 1 above.

The next step, involving behavioral decomposition, must now be performed. Each of the following behavioral layers was implemented, tested, and debugged in turn:

1. Standup: Clearly, before the robot can walk, it needs to lift its body off of the ground. Further decomposition leads to the development of two AFSMs, one to control the leg's swing position and the other its lift. When all six legs operate under the standup behavior, the robot assumes a stance from which it can begin walking.
2. Simple walk: This requires that the leg be lifted off the ground and swung forward (advance). A variety of sensor data is used to coordinate the motion between legs, including encoders returning the position of each leg's joints. When appropriately coordinated, a simple walk over smooth terrain (tripod gait) is achieved.
3. Force balancing: Now the issues concerning rough terrain are confronted. Force sensors are added to the legs, providing active compliance to changes in the ground's contour.

4. Leg lifting: This helps with stepping over obstacles. When required, the leg can lift itself much higher than normal to step over obstacles.
5. Whiskers: These sensors are added to anticipate the presence of an obstacle rather than waiting for a collision. This capability emerges as important through experiments with the previous behaviors.
6. Pitch stabilization: Further experiments show that the robot tends to bump into the ground either fore or aft (pitching). An inclinometer is added to measure the robot's pitch and use it to compensate and prevent bumping. Now the robot's walking capabilities are complete.
7. Prowling: The walking robot is now concerned with moving toward a detected human. The infrared sensors are tied in. When no person is present, walking is suppressed. As soon as someone steps in front of the robot, the suppression stops and walking begins.
8. Steered prowling: The final behavior allows the robot to turn toward the person in front of it and follow him. The difference in readings between two IR sensors is used to provide the stimulus, and the swing end points for the legs on each side of the robot are determined by the difference in strength.

The completed robot, satisfying the task criteria established for it, consists of fifty-seven AFSMs built in an incremental manner. Each layer has been tested experimentally before moving onto the next, and the results of those tests have established the need for additional layers (e.g., whiskers and pitch stabilization).

4.3.4 Foraging Example

The foraging example presented earlier also illustrates subsumption-based design. In particular, the robots Mataric (1993a) constructed for several tasks including foraging provide the basis for this discussion. The robots are programmed in the Behavior Language. The target hardware is an IS Robotics system (figure 4.6).

Each behavior in the system is encoded as a set of rules (standard for the Behavior Language). The overall system has actually been developed as a multiagent robotic system (chapter 9), but for now we will restrict this discussion to a single robot foraging. The following behaviors are involved:

- Wandering: move in a random direction for some time.
- Avoiding:
  - turn to the right if the obstacle is on the left, then go.
  - turn to the left if the obstacle is on the right, then go.

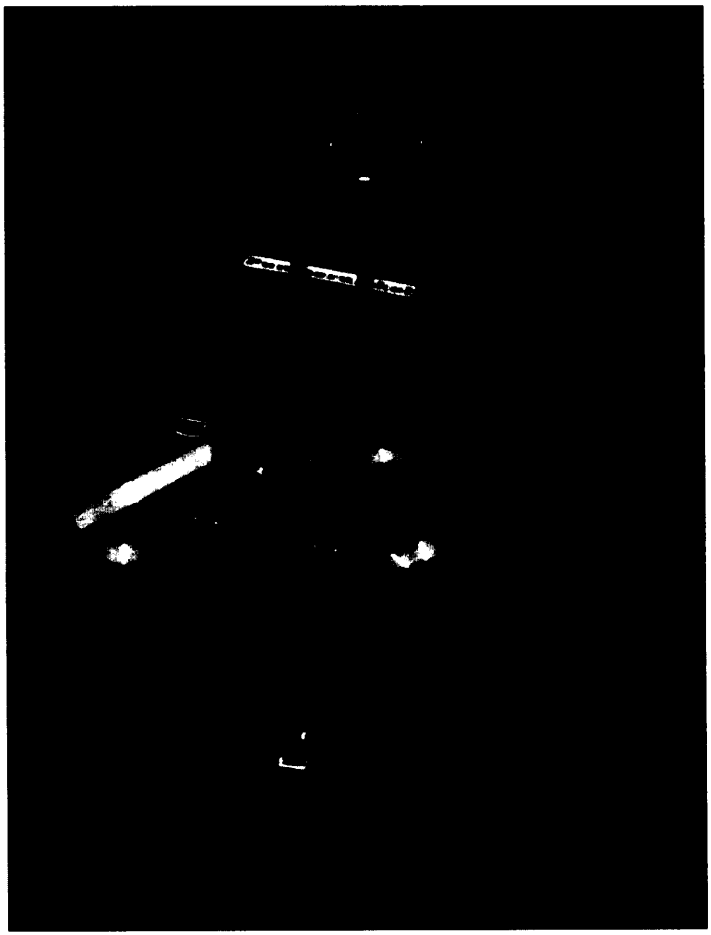


Figure 4.6
Subsumption-based foraging robot: R1. (Photograph courtesy of IS Robotics, Somerville, MA.)

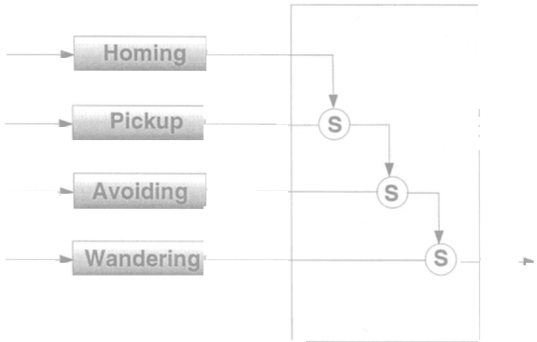


Figure 4.7
SR diagram for subsumption-based foraging robot.

- after three attempts, back up and turn.
- if an obstacle is present on both sides, randomly turn and back up.

■ Pickup: Turn toward the sensed attractor and go forward. If at the attractor, close gripper.

■ Homing: Turn toward the home base and go forward, otherwise if at home, stop.

Figure 4.7 illustrates the SR diagram for this set of behaviors. Priority-based arbitration is the coordination mechanism, and the robot is executing only one behavioral rule at any time. Note in particular that when the robot senses the attractor, wandering is suppressed and when the attractor is grabbed, homing then suppresses pickup (allowing the robot to ignore the potential distraction of other attractors it might encounter along the way).

4.3.5 Evaluation

When the criteria presented in section 4.1.3, are applied to evaluate the subsumption architecture, they identify the following strengths:

■ Hardware retargetability: Subsumption can compile down directly onto programmable-array logic circuitry (Brooks 1987b).

■ Support for parallelism: Each behavioral layer can run independently and asynchronously.

■ Niche targetability: Custom behaviors can be created for specific task-environment pairs.

The following characteristics emerge as neither strength nor weaknesses:

- Robustness: This can be successfully engineered into these systems but is often hard-wired (Ferrell 1994) and hence hard to implement.
- Timeliness for development: Some support tools exist for these systems, but a significant learning curve is still associated with custom behavioral design. Experimental design, involving trial-and-error development, can slow development. Also, consistent with Brooks' philosophy, simulators are not used to pretest behavioral efficiency.

Under the criteria, the following show up as weaknesses:

- Run time flexibility: The priority-based coordination mechanism, the ad hoc flavor of behavior generation, and the architecture's hard-wired aspects limit the ways the system can be adapted during execution.
- Support for modularity: Although behavioral reuse is possible through the Behavior Language, it is not widely evidenced in constructed robots. Subsumption has also been criticized on the basis that since upper layers interfere with lower ones, they cannot be designed completely independently (Hartley and Pipitone 1991). Also behaviors cannot always be prioritized (nor should they be), leading to artificial arbitration schemes (Hartley and Pipitone 1991). Commitment to subsumption as the sole coordination mechanism is restrictive.

4.3.6 Subsumption Robots

Many different robots (figure 4.8) have been constructed using the subsumption architecture. Brooks 1990b reviews many of them. They include

- Allen: the first subsumption-based robot, which used sonar for navigation based on the ideas in Brooks 1986.
- Tom and Jerry: two small toy cars equipped with infrared proximity sensors (Brooks 1990b).
- Genghis and Attila: six-legged hexapods capable of autonomous walking (Brooks 1989a).
- Squirt: a two-ounce robot that responds to light (Flynn et al. 1989).
- Toto: the first map-constructing, subsumption-based robot and the first to use the Behavior Language (Mataric 1992b).
- Seymour: a visual motion-tracking robot (Brooks and Flynn 1989).
- Tito: a robot with stereo navigational capabilities (Sarachik 1989).
- Polly: a robotic tour guide for the MIT AI lab (Horswill 1993b).
- Cog: a robot modeled as a humanoid from the waist up, and used to test theories of robot-human interaction and computer vision (Brooks and Stein 1994).

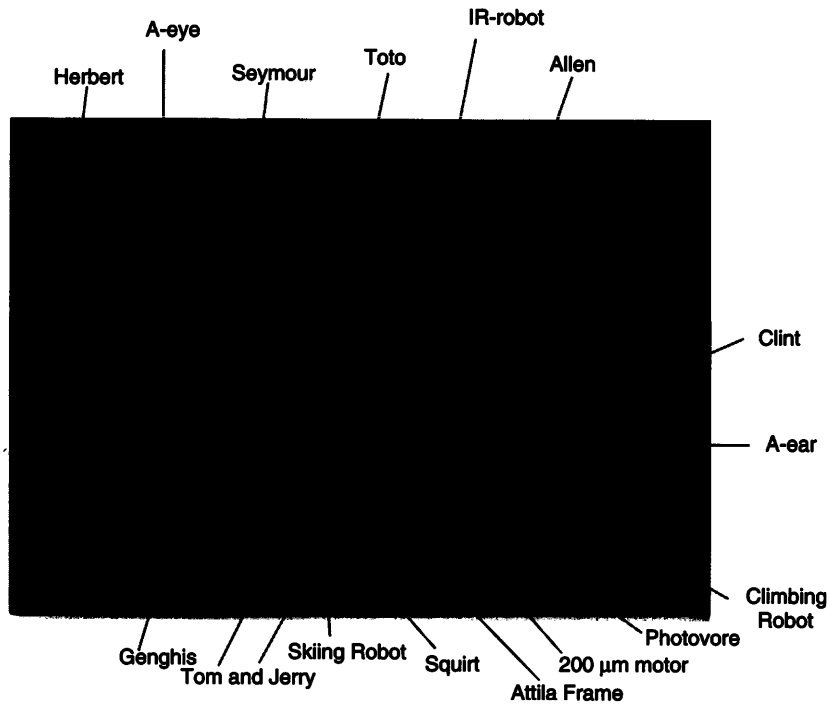


Figure 4.8
Robots of the MIT Mobile Robot Laboratory. (Photograph courtesy of Rodney Brooks.)

4.4 MOTOR SCHEMAS

Another approach, more strongly motivated by the biological sciences, appeared on the heels of the subsumption architecture. This behavior-based method used schema theory, which we reviewed in chapter 2. We recall from that review that schema theory provides the following capabilities for specifying and designing behavior-based systems (adapted from Arbib 1992):

- Schema theory explains motor behavior in terms of the concurrent control of many different activities.
- A schema stores both how to react and the way that reaction can be realized.
- Schema theory is a distributed model of computation.
- Schema theory provides a language for connecting action and perception.
- Activation levels are associated with schemas that determine their readiness or applicability for acting.

- Schema theory provides a theory of learning through both schema acquisition and schema tuning.
- Schema theory is useful for explaining the brain's functioning as well as distributed AI applications (such as behavior-based robotics).

Schema theory is an attempt to account for the commonalities in both neurobiological and artificial behavior, and Arkin chose it as a suitable vehicle to implement robotic behavior.

Arkin (1989a, 1990a, 1993) addressed the implications of schema theory for autonomous robotics:

1. Schemas provide large grain modularity, in contrast to neural network models, for expressing the relationships between motor control and perception.
2. Schemas act concurrently as individual distributed agents in a cooperative yet competing manner and thus are readily mappable onto distributed processing architectures.
3. Schemas provide a set of behavioral primitives by which more complex behaviors (assemblages) can be constructed.
4. Cognitive and neuroscientific support exists for the underpinnings of this approach. These can be modified, if appropriate, as additional neuroscientific or cognitive models become available.

The overall method of schema-based robotics is to provide behavioral primitives that can act in a distributed, parallel manner to yield intelligent robotic action in response to environmental stimuli. Lyons has also used schema theory in his research (sections 3.2.4.1 and 6.6.3), but here we focus on the methodology Arkin adopted.

The motor schema method differs from other behavioral approaches in several significant ways:

- Behavioral responses are all represented in a single uniform format: vectors generated using a potential fields approach (a continuous response encoding).
- Coordination is achieved through cooperative means by vector addition.
- No predefined hierarchy exists for coordination; instead, the behaviors are configured at run-time based on the robot's intentions, capabilities, and environmental constraints. Schemas can be instantiated or deinstantiated at any time based on perceptual events, hence the structure is more of a dynamically changing network than a layered architecture.
- Pure arbitration is not used; instead, each behavior can contribute in varying degrees to the robot's overall response. The relative strengths of the behaviors (**G**) determine the robot's overall response.

Table 4.2
Motor schemas

Name	Motor Schemas
Background	Reactive component of AuRA Architecture
Precursors	Arbib 1981; Khatib 1985
Principal design method	Ethologically guided
Developer	Ronald Arkin (Georgia Tech)
Response encoding	Continuous using potential field analog
Coordination method	Cooperative via vector summation and normalization
Programming method	Parameterized behavioral libraries
Robots fielded	HARV, George, Ren and Stimpy, Buzz, blizzards, mobile manipulator, others
References	Arkin 1987a; Arkin 1989b; Arkin 1992a

- Perceptual uncertainty can be reflected in the behavior's response by allowing it to serve as an input within the behavioral computation.

Table 4.2 summarizes the important aspects of this architecture. The remainder of this section studies the details of its implementation.

4.4.1 Schema-Based Behaviors

Motor schema behaviors are relatively large grain abstractions reusable over a wide range of circumstances. Many of the behaviors have internal parameters that provide additional flexibility in their deployment. The behaviors generally are analogous to animal behaviors (section 2.4), at least those useful for navigational tasks.

A perceptual schema is embedded within each motor schema. These perceptual schemas provide the environmental information specific for that particular behavior. Perception is conducted on a need-to-know basis: individual perceptual algorithms provide the information necessary for a particular behavior to react. Chapter 7 details this sensing paradigm, referred to as *action-oriented perception*. Suffice it to say for now that attached to each motor schema is a perceptual process capable of providing suitable stimuli, if present, as rapidly as possible. Perceptual schemas are recursively defined, that is, perceptual subschemas can extract pieces of information that are subsequently processed by another perceptual schema into a more behaviorally meaningful unit. An example might involve recognition of a person with more than one sensor: Infrared

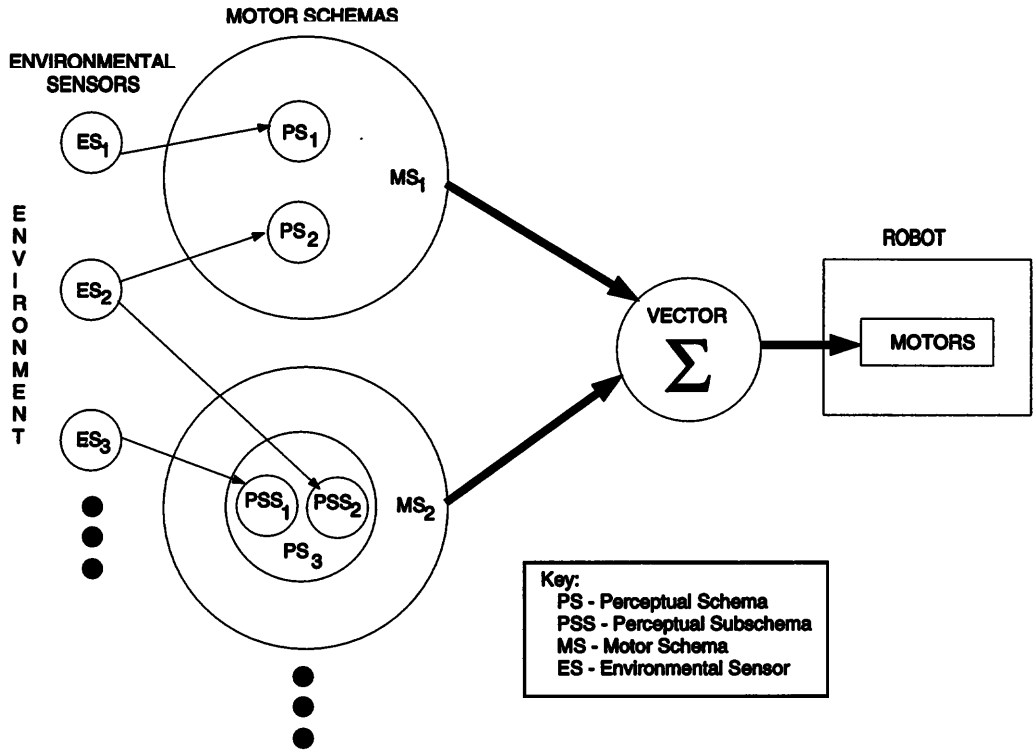


Figure 4.9
Perception-action schema relationships.

sensors can provide a heat signature, whereas computer vision may provide a human shape. The information generated from each of these lower-level perceptual processes would be merged into a higher-level interpretation before passing it on to the motor behavior. This enables the use of multiple sensors within the context of a single sensorimotor behavior. Figure 4.9 illustrates this relationship.

Each motor schema has as output an action vector (consisting of both orientation and magnitude components) that defines the way the robot should move in response to the perceived stimuli. This approach has been used for ground-based navigation where each vector is two dimensional (Arkin 1989b), for generating three-dimensional vectors for use in flying or underwater navigation (Arkin 1992a), and for use in mobile manipulators with many additional degrees of freedom (Cameron et al. 1993).

Many different motor schemas have been defined, including

- Move-ahead: move in a particular compass direction.
- Move-to-goal: move towards a detected goal object. Two versions exist of this schema: ballistic and controlled.
- Avoid-static-obstacle: move away from passive or nonthreatening navigational barriers.
- Dodge: sidestep an approaching ballistic projectile.
- Escape: move away from the projected intercept point between the robot and an approaching predator.
- Stay-on-path: move toward the center of a path, road, or hallway. For three-dimensional navigation, this becomes the stay-in-channel schema.
- Noise: move in a random direction for a certain amount of time.
- Follow-the-leader: move to a particular location displaced somewhat from a possibly moving object. (The robot acts as if it is leashed invisibly to the moving object.)
- Probe: move toward open areas.
- Dock: approach an object from a particular direction.
- Avoid-past: move away from areas recently visited.
- Move-up, move-down, maintain-altitude: move upward or downward or follow an isocontour in rough terrain.
- Teleautonomy: allows human operator to provide internal bias to the control system at the same level as another schema.

These are the basic building blocks for autonomous navigation within this architecture. Figure 4.10 depicts several of the schemas. Remember that although the entire field is illustrated, only a single vector needs to be computed at the robot's current location. This ensures extremely fast computation.

The actual encodings for several schemas appear below, where $V_{magnitude}$ denotes the magnitude of the resultant response vector and $V_{direction}$ represents its orientation:

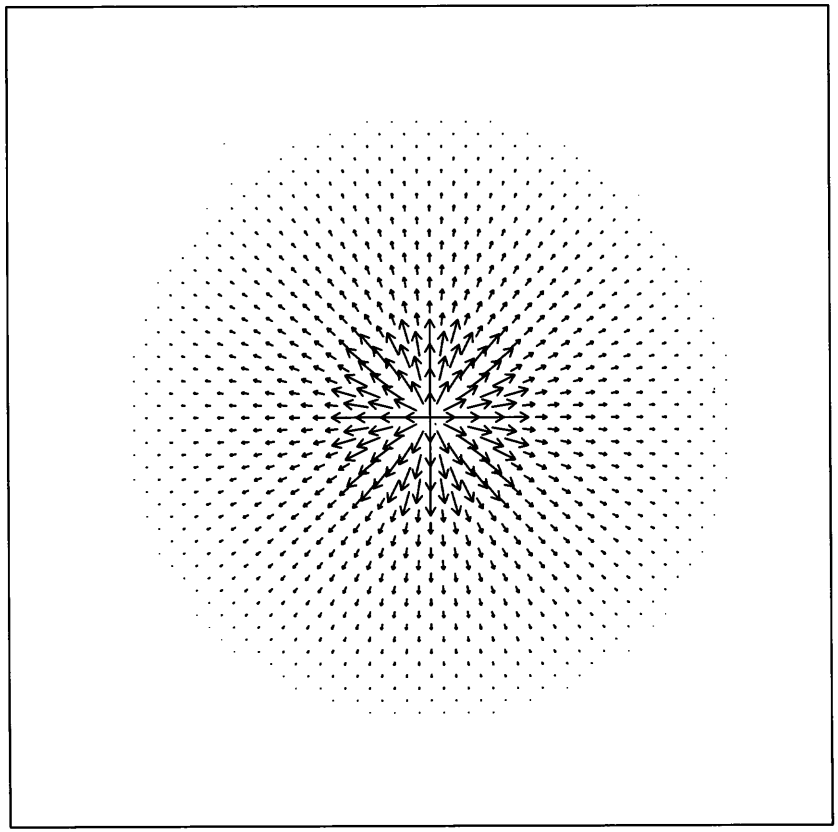
- move-to-goal (ballistic):

$V_{magnitude}$ = fixed gain value.

$V_{direction}$ = towards perceived goal.

- avoid-static-obstacle:

$$V_{magnitude} = \begin{cases} 0 & \text{for } d > S \\ \frac{S-d}{S-R} * G & \text{for } R < d \leq S \\ \infty & \text{for } d \leq R \end{cases},$$



(A)

Figure 4.10
Representative schemas: (A) avoid-static-obstacle, (B) move-ahead (toward east), (C) move-to-goal, guarded (compare to ballistic version in figure 3.16 (B)), (D) noise, (E) stay-on-path, and (F) dodge.

where

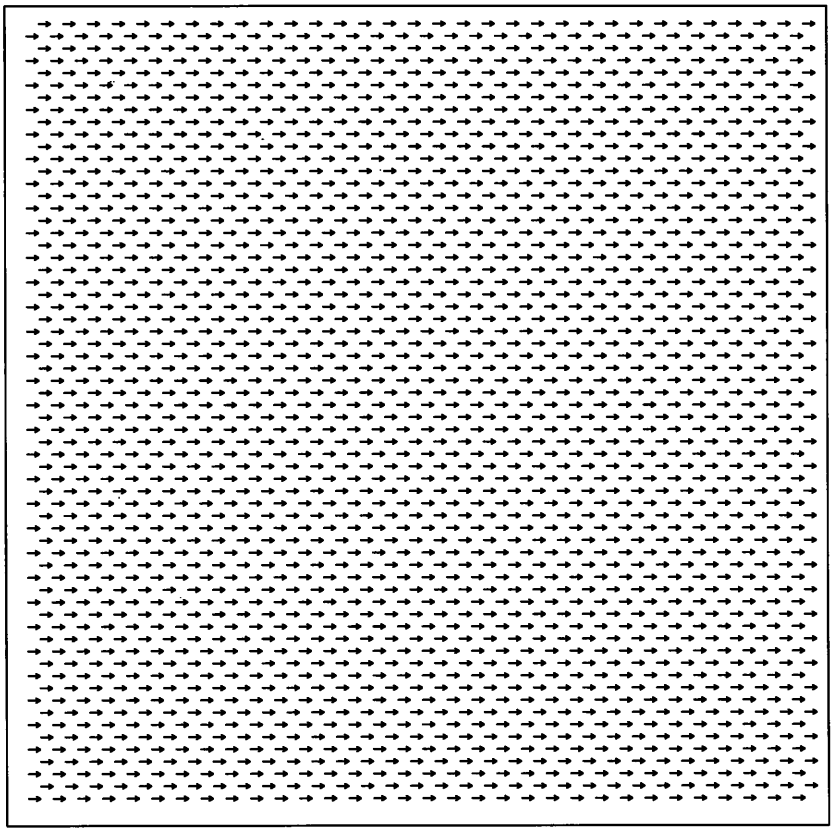
S = sphere of influence (radial extent of force from the center of the obstacle),

R = radius of obstacle,

G = gain, and

d = distance of robot to center of obstacle.

$V_{direction}$ = radially along a line from robot to center of obstacle, directed away from the obstacle.



(B)

Figure 4.10 ***(continued)***

- stay-on-path

$$V_{magnitude} = \begin{cases} P & \text{for } d > (W/2) \\ \frac{d}{W/2} * G & \text{for } d \leq (W/2) \end{cases},$$

where

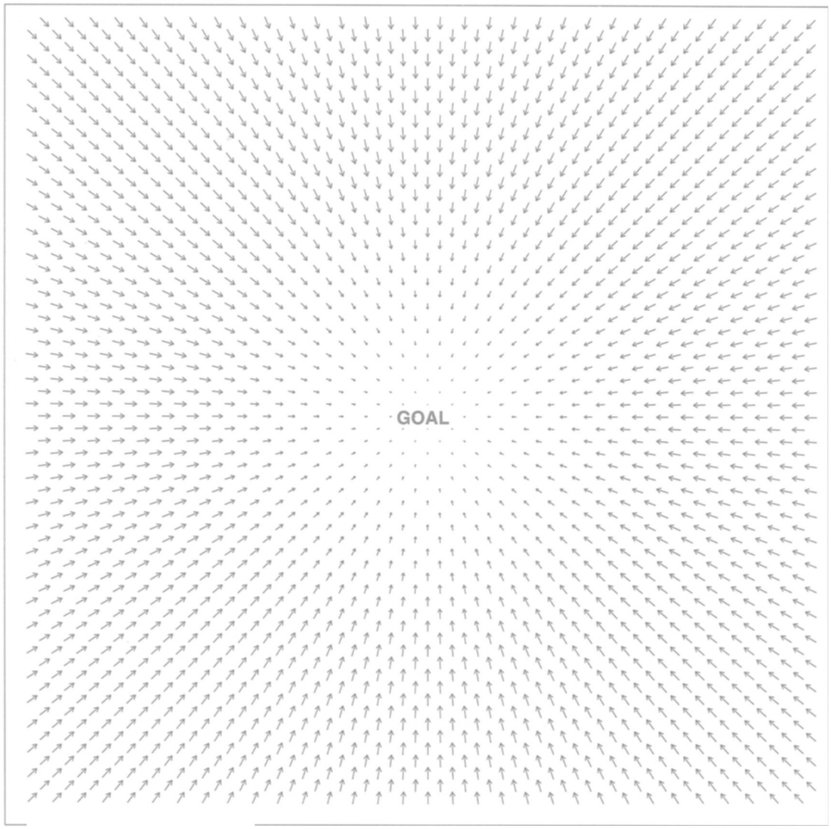
W = width of path,

P = off-path gain,

G = on-path gain, and

d = distance of robot to center of path.

$V_{direction}$ = along a line from robot to center of path, heading toward centerline.



(C)

Figure 4.10 ***(continued)***

- move-ahead

$V_{magnitude}$ = fixed gain value.

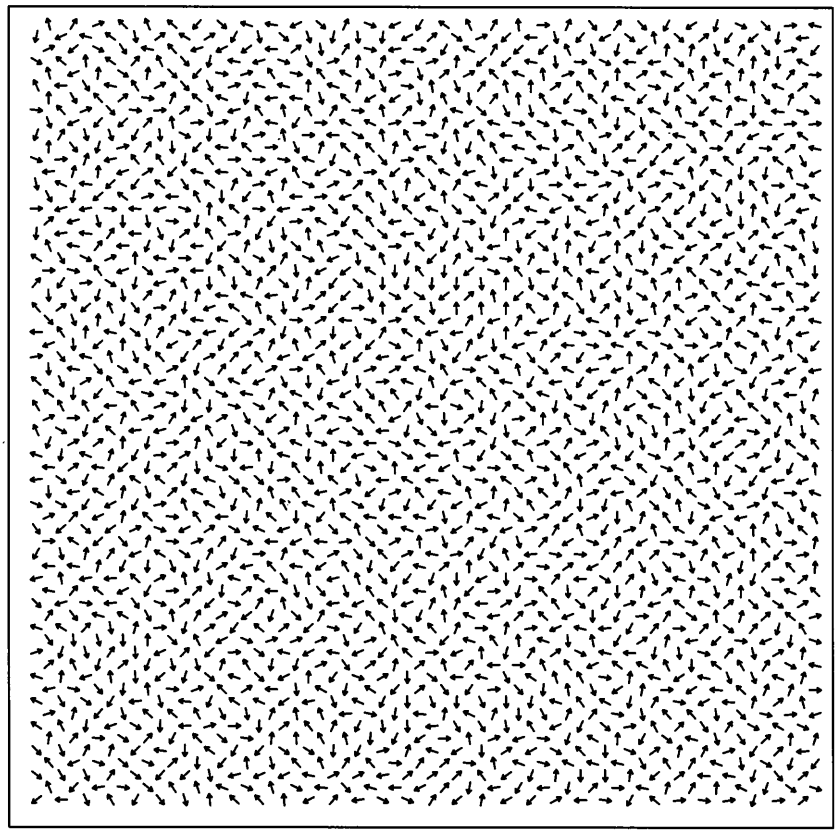
$V_{direction}$ = specified compass direction.

- noise

$V_{magnitude}$ = fixed gain value.

$V_{direction}$ = random direction changed every p time steps (p denotes persistence).

It can be seen that the actual response computations are very simple and fast.



(D)

Figure 4.10 ***(continued)***

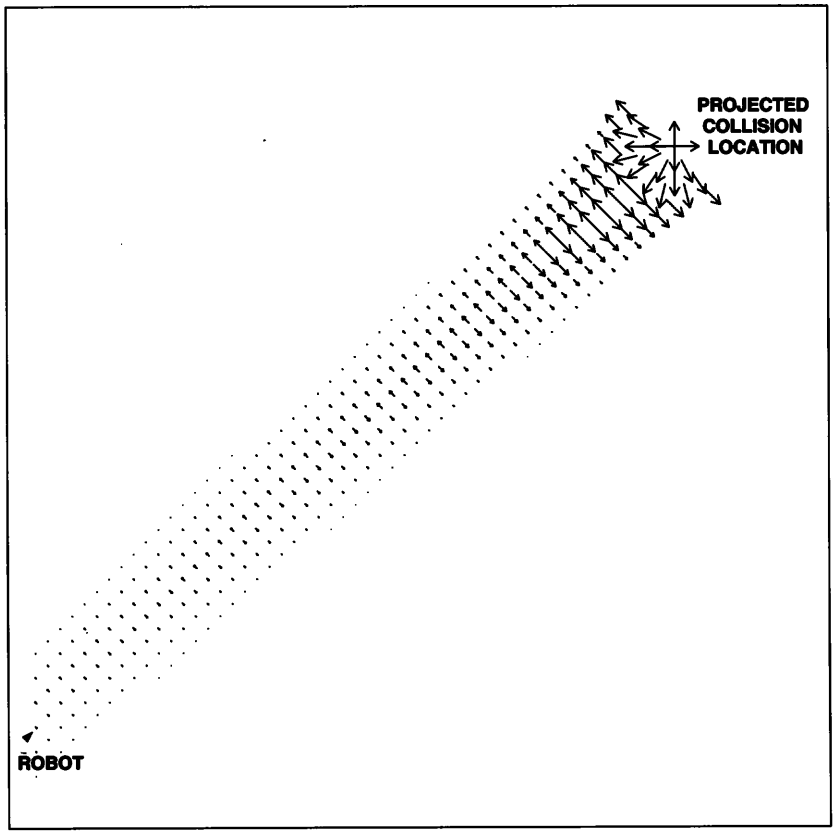
4.4.2 Schema-Based Coordination

The next issue is how coordination is accomplished with motor schemas. The answer is straightforward: vector summation. All active behaviors contribute to some degree to the robot's global motion. G, the gain vector, determines the relative composition for each behavior. The notion of schema gains is loosely aligned with the concept of activation levels mentioned previously. In a system where no learning or adaptation is permitted, the gain levels remain constant throughout the run. We will see in chapter 8 that this can be modified during execution to permit learning. Action-selection techniques, soon to be described in more detail, also reflect the notion of activation levels.

(E)

Figure 4.10 ***(continued)***

Returning to coordination, each schema output vector is multiplied by its associated gain value and added to all other output vectors. The result is a single global vector. That vector must be normalized in some manner (often merely by clipping the magnitude) to ensure that it is executable on the robot. The resulting normalized global vector is sent to the robot for execution. This sense-react process is repeated as rapidly as possible. The schemas can operate asynchronously, each delivering its data as quickly as it can. Perceptual performance generally limits overall processing speed since the simple and distributed motor response computation is extremely rapid. Some care must be taken to ensure that the reaction produced is based on information still relevant (i.e., the perceptual data is not too old). Normally, the perceptual algorithms'



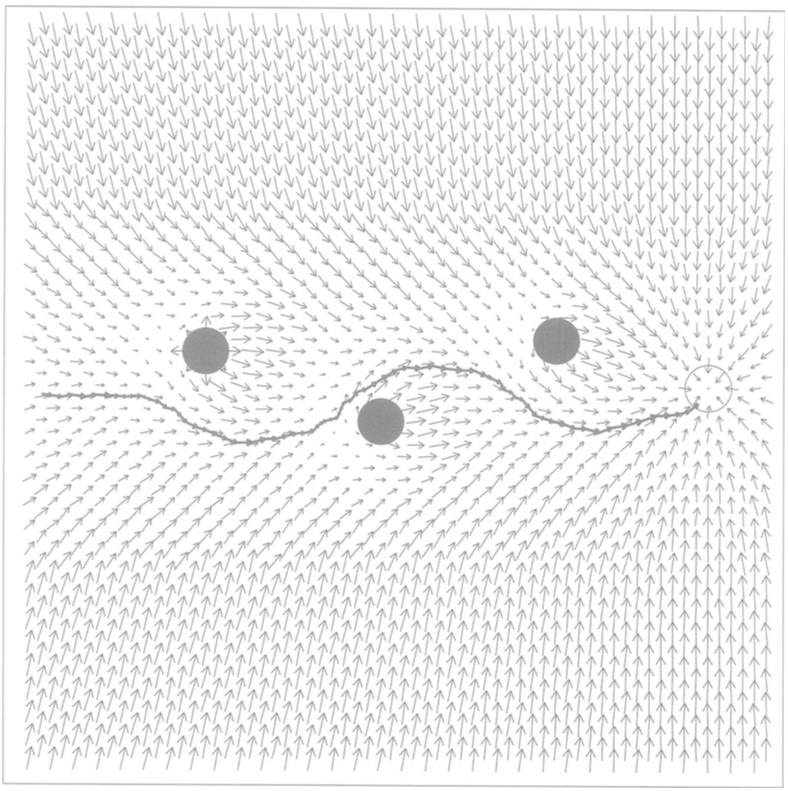
(F)

Figure 4.10 ***(continued)***

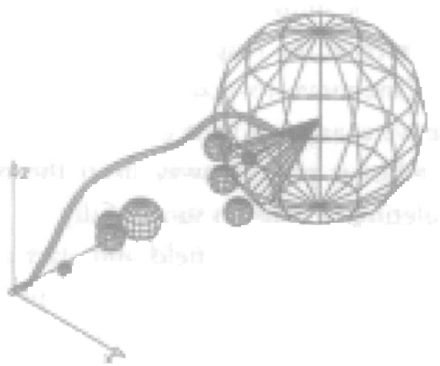
action-oriented design ensures relatively prompt processing independent of the data source.

Figure 4.11 shows several different types of robot paths resulting from these methods. It is interesting to observe some of the biological parallels for these type of systems (figure 4.12).

Section 3.3.2 described certain problems endemic to the use of potential fields, in particular local minima and cyclic behavior. Schema-based systems are not immune to these problems, nor have they been ignored. One of the simplest methods to address these problems is through noise, injecting randomness into the behavioral system through the noise schema. Noise is a common technique used to deal with local minima in many gradient descent methods, for

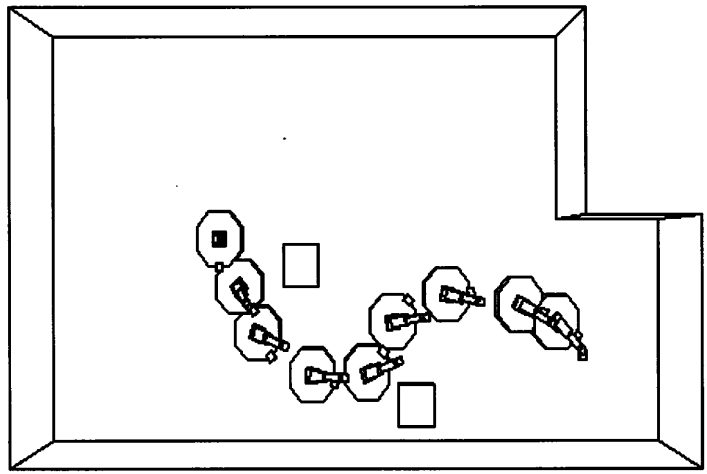


(A)



(B)

Figure 4.11



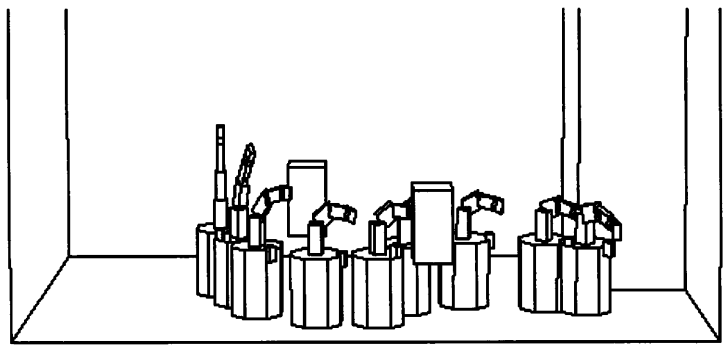
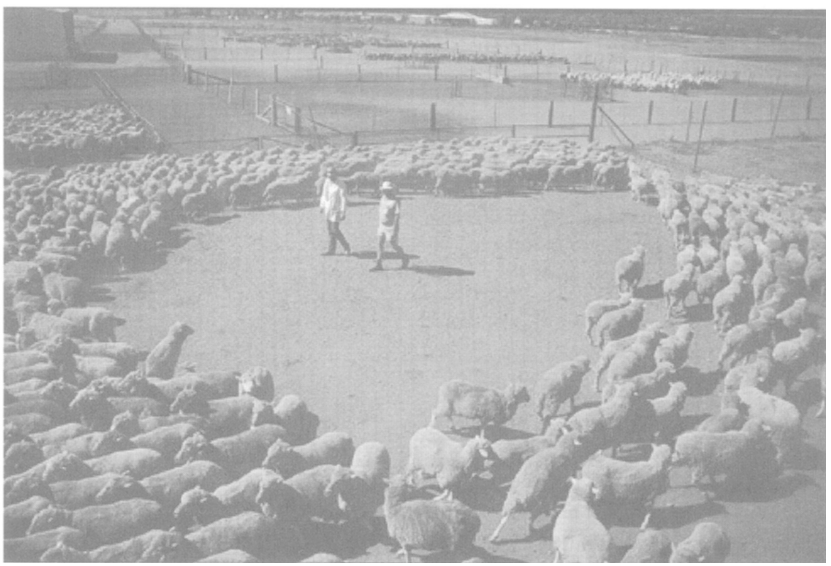


Figure 4.11 ***(continued)***
Representative simulated robot paths: (A) stay-on-path, move-to-goal, 3 avoid-static-obstacle, and noise schemas; (B) three-dimensional docking and avoid-static-obstacles; and (C) mobile manipulator moving to target object through obstacles.



(B)

Figure 4.12

example, simulated annealing (Hinton and Sejnowski 1986) and mutation in genetic algorithms (Goldberg 1989), to name a few. In schema-based control, randomness works in several ways: in some cases it prevents the entry into local minima, acting as a sort of "reactive grease" (figure 4.13). In general, it is always useful to inject a small amount of noise into a schema-based system to help ensure progress.

Balch and Arkin (1993) developed another schema, avoid-past, to ensure progress is made even if the robot tends to stall. Avoid-past uses a short-term representation as it retains a timewindow into the past indicating where the robot has been recently. It is still reactive, however, since no path planning for the robot is ever conducted, and the output of this behavior is a vector of the same form as all the other behaviors and is combined in the same way. Repulsive forces are generated from recently visited areas that prevent the robot from stalling when not at its goal. This approach has proven very effective in even degenerate cases (figure 4.14).

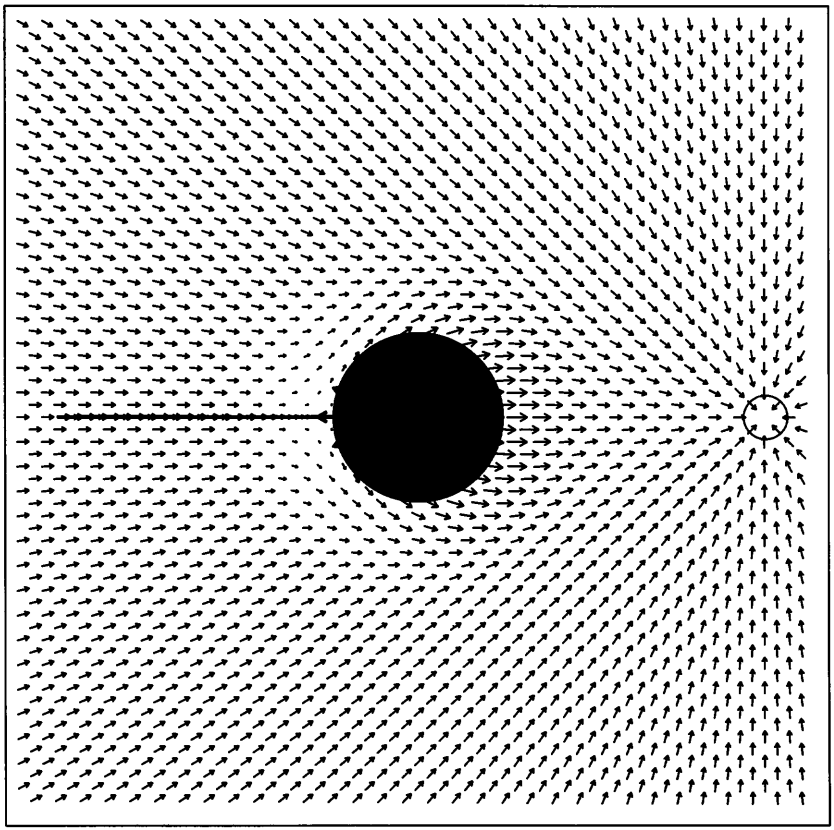
Additionally chapter 8 will discuss several adaptive and learning techniques that have been applied to improve navigational performance.

4.4.3 Design in Motor Schema–Based Systems

The design process for building a schema-based robotic system is typically as follows:

1. Characterize the problem domain in terms of the motor behaviors necessary to accomplish the task.
2. Decompose the motor behaviors to their most primitive level, using biological studies whenever feasible for guidelines.
3. Develop formulas to express the robot's reaction to perceived environmental events.
4. Conduct simple simulation studies assessing the desired behaviors' approximate performance in the proposed environment.
5. Determine the perceptual requirements needed to satisfy the inputs for each motor schema.

Figure 4.12 ***(continued)***
Biological parallels: (A) A school of anchovies being attacked by diving birds. (Photograph courtesy of Gary Bell). (B) A herd of sheep in flight from their handlers. (Photograph courtesy of Temple Grandin, Colorado State University.)

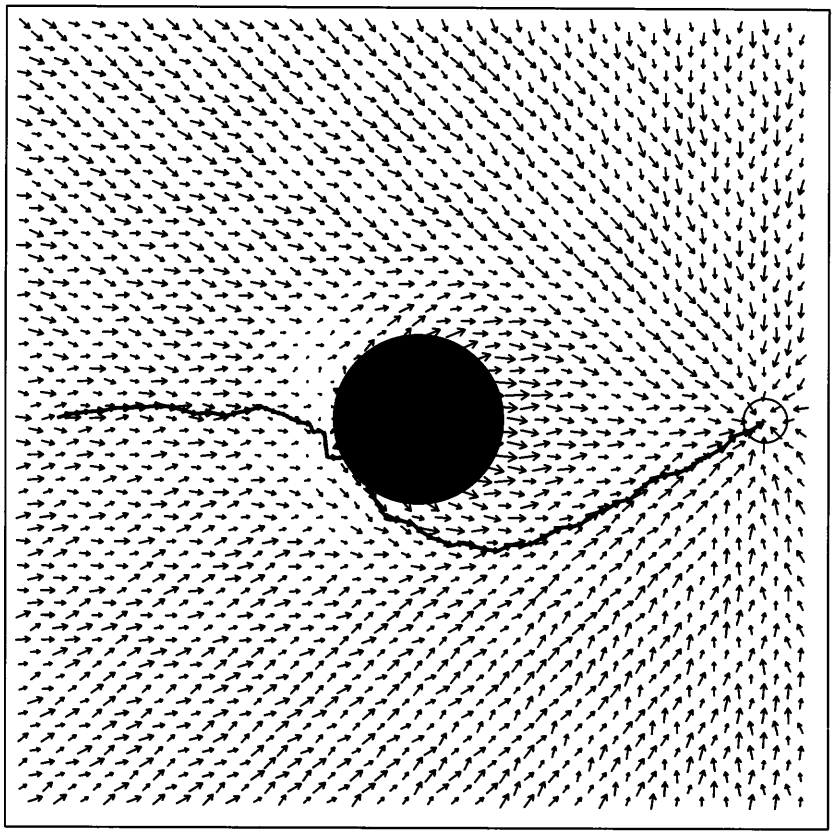


(A)

Figure 4.13
Effect of noise on schema-based navigation (A) Exact counterbalancing of forces causes robot to stall; (B) With noise added, the problem is prevented.

6. Design specific perceptual algorithms that extract the required data for each behavior, utilizing action-oriented perception, expectations, and focus-of-attention techniques to ensure computational efficiency (chapter 7).
7. Integrate the resulting control system onto the target robot.
8. Test and evaluate the system's performance.
9. Iterate and expand behavioral repertoire as necessary.

Behavioral software reuse greatly simplifies this process, because schemas developed using biological guidelines often have extensive utility in many different circumstances.



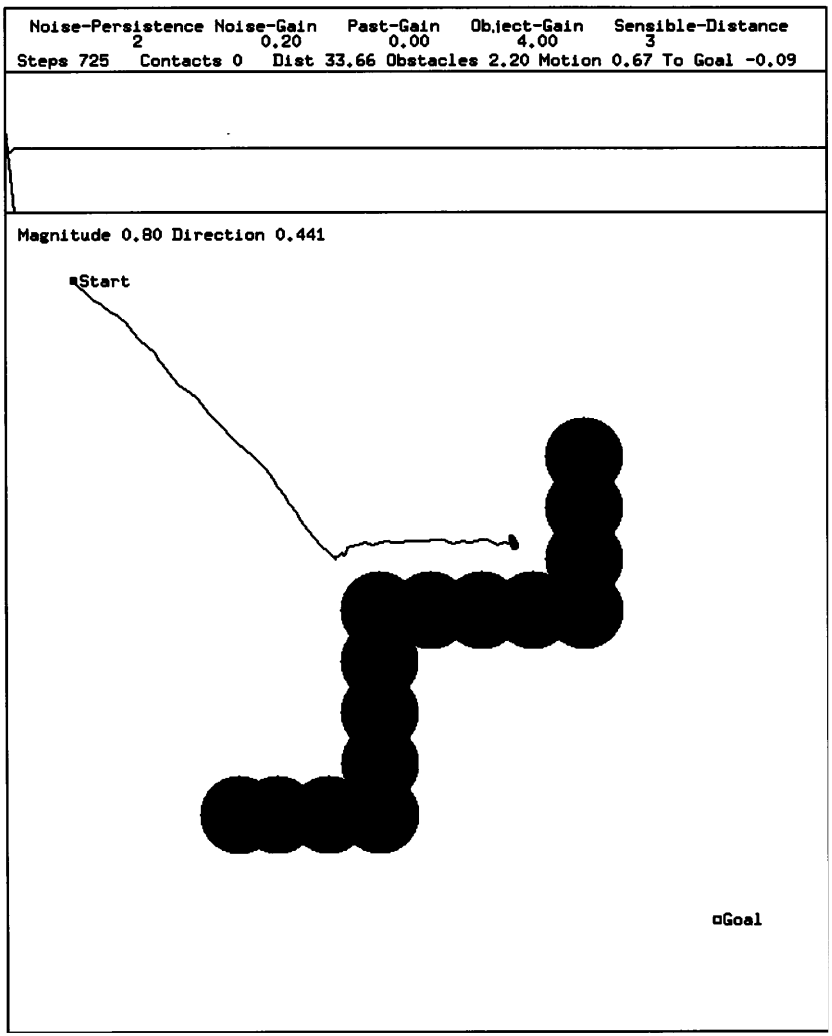
(B)

Figure 4.13 ***(continued)***

4.4.4 Foraging Example

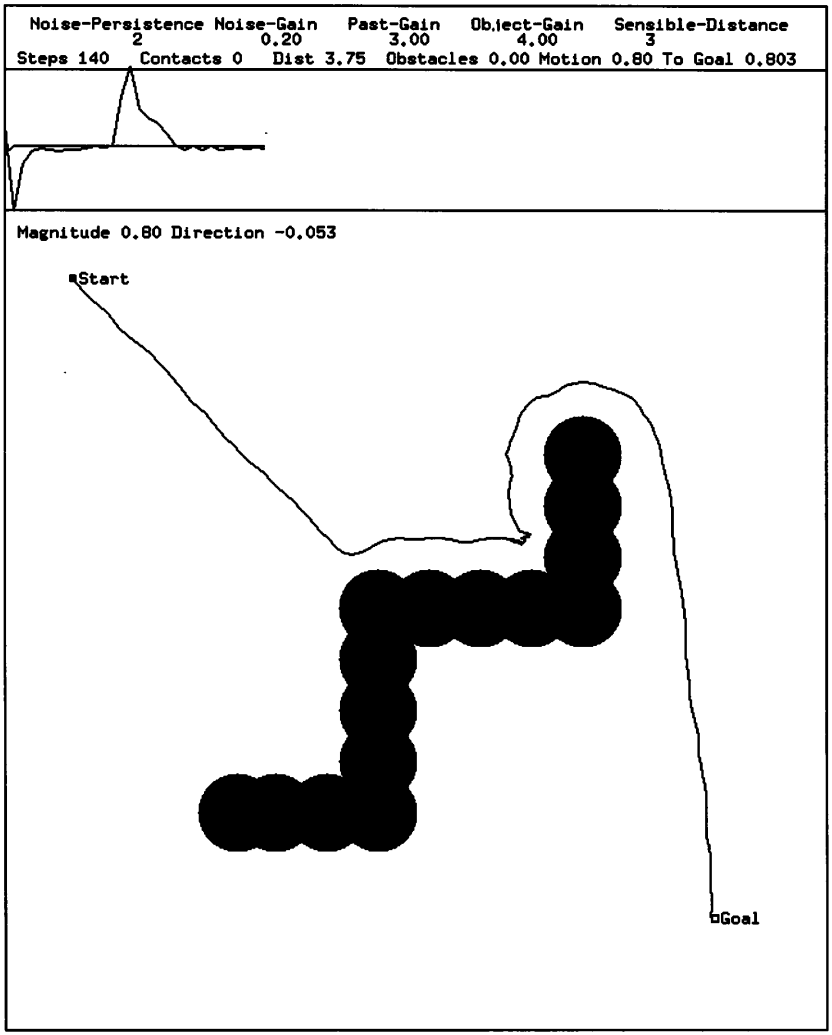
Motor schemas have been used in several implementations of foraging systems. The first example, mirroring the FSA shown in figure 4.2, consists of three assemblages, each consisting of up to four behaviors (Arkin 1992b). The primitive behaviors are

- Avoid-static-obstacle: instantiated differently for either environmental obstacles or other robots.
- Move-to-goal: changes its attention from the attractor to the home base depending upon the state in which the robot finds itself.



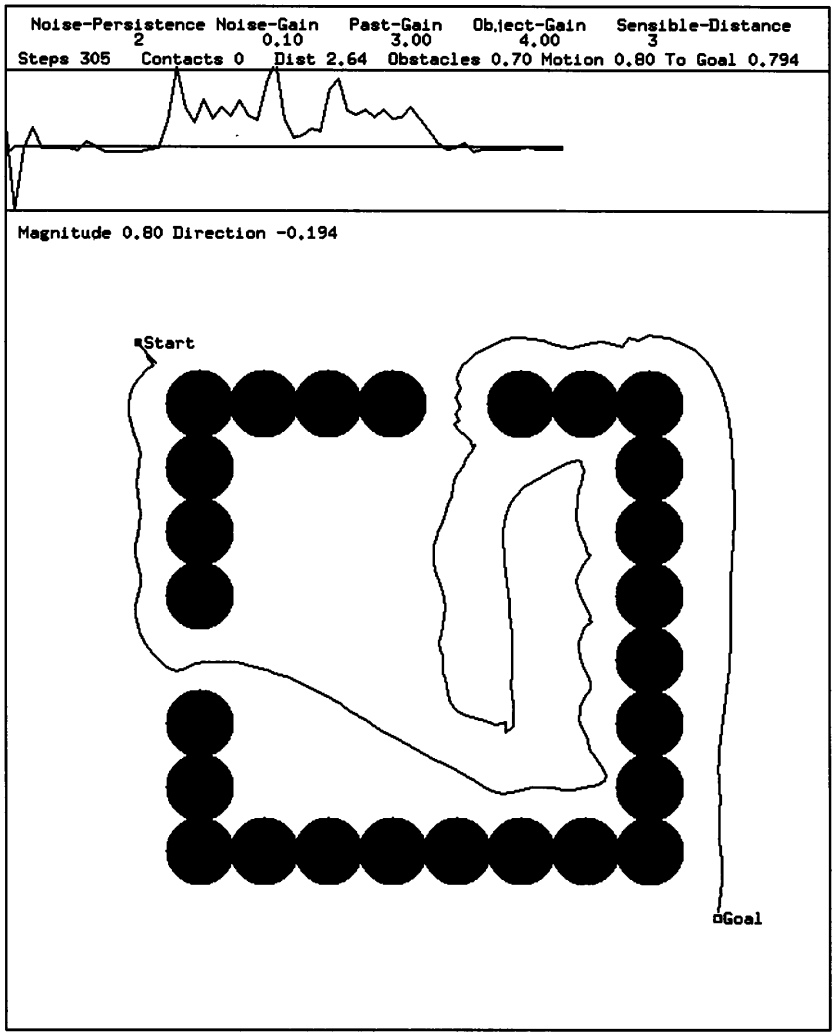
(A)

Figure 4.14
Avoid-past schema usage for overcoming local minima. Repulsive forces are generated at recently visited areas, preventing the robot from stalling: (A) without and (B) with *avoid-past* behavior; (C) deep box with *avoid-past*; (D) maze with *avoid-past*.



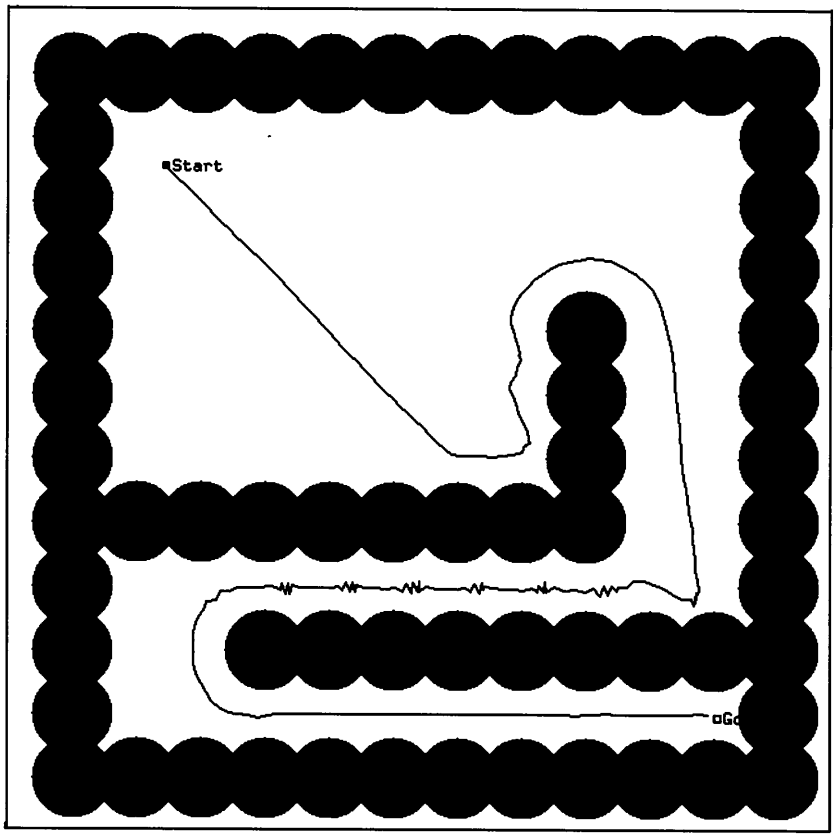
(B)

Figure 4.14 ***(continued)***



(C)

Figure 4.14 ***(continued)***



(D)

Figure 4.14 ***(continued)***

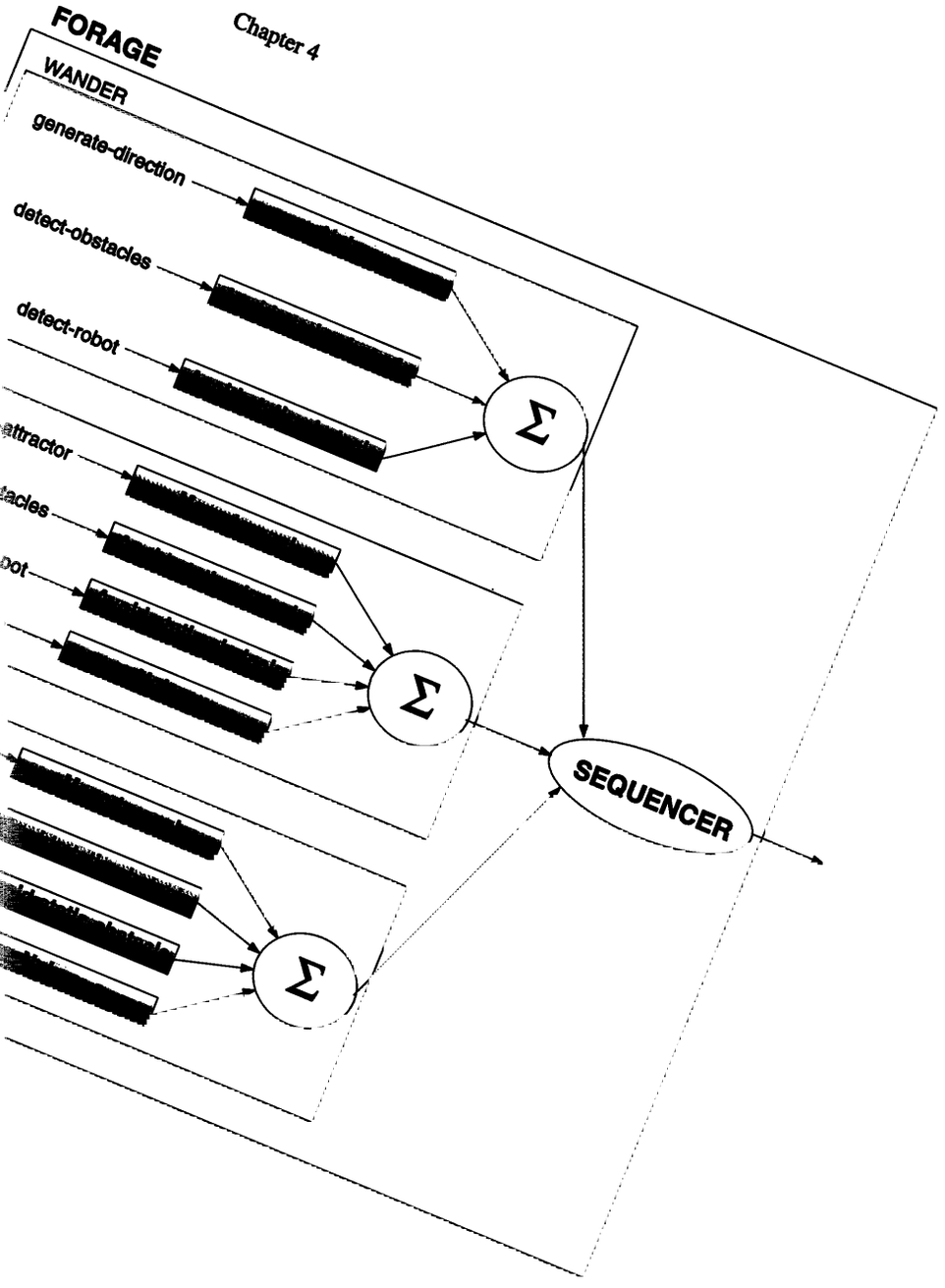
- Noise: Initially set at a high gain to ensure broad exploration of the area, then reduced greatly upon encountering an attractor.

Figure 4.15 depicts the behavioral configuration using an SR diagram.

Subsequently, Balch et al. (1995) created more-complex foraging robots (figure 4.16) using a similar methodology for use in a robot competition. Chapter 9 discusses this multirobot implementation in more detail.

4.4.5 Evaluation

When evaluated using the criteria presented in section 4.1, motor schema–based robotic systems are found to have the following strengths:


FORAGE
WANDER
generate-direction
detect-obstacles
detect-robot
Σ
ttractor
acles
ot
Σ
Σ
SEQUENCER

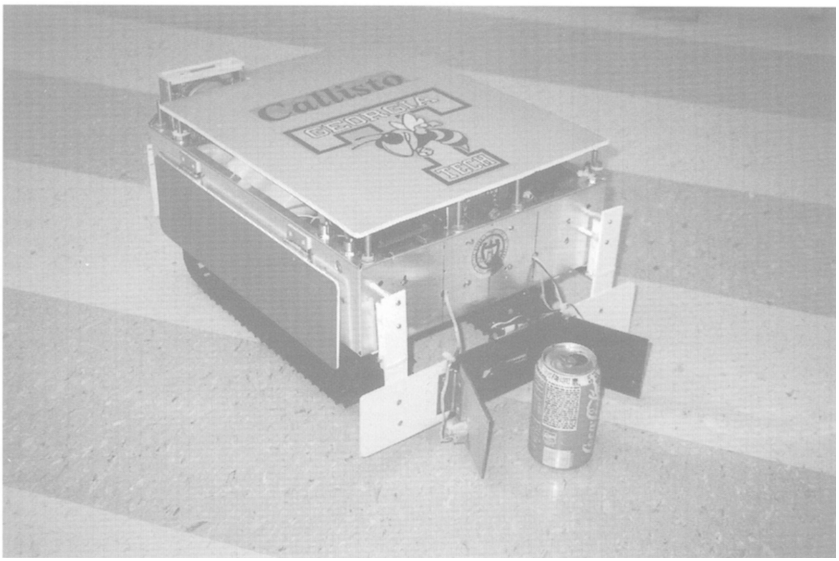


Figure 4.16
Callisto, a foraging schema-based robot.

- Support for parallelism: Schema theory is a distributed theory of computation involving multiple parallel processes. Motor schemas are naturally parallelizable.
- Run time flexibility: As schemas are software agents, instantiated at run time as processes and are not hard-wired into the architecture, it is simple to reconfigure the behavioral control system at any time.
- Timeliness for development and support for modularity: Schemas are essentially software objects and are by definition modular. They can be stored in behavioral libraries and are easily reused (Mackenzie, Cameron, and Arkin 1995).

The following is found to be neither a strength nor a weakness:

- Robustness: As with any reactive system, schemas can well cope with change in the environment. One deficiency lies in the use of potential field analogs for behavioral combination, which has several well-known problems. Specific methods, however, such as the introduction of noise and the *avoid-past* behavior, have been developed to circumvent this difficulty.

The following weaknesses are identified under the evaluation criteria:

Figure 4.17
Several motor schema-based robots. First row (left to right): Io, Callisto, and Ganymede. Second row: Shannon and Sally. Third row: George, Ren, and Stimpy. Rear: GT Hummer.

- Niche targetability: Although it is feasible to design niche robots, the generic modular nature of the primitive schemas somewhat discourages the design of very narrowly focused components.
- Hardware retargetability: Schema-based systems are essentially software architectures mappable onto hardware multiprocessor systems. They do not provide the hardware compilers that either subsumption or Gapps does. Hardware mappings are feasible, however (Collins, Arkin, and Henshaw 1993), just not as convenient as with some other systems.

4.4.6 Schema-Based Robots

The incremental development of the library of motor schemas can be traced through a series of fielded mobile robots (figure 4.17). Schemas from earlier robots were easily reused for newer machines as they became available.

- HARV: An early Denning mobile robot, named after the cocktail Harvey Wallbanger, which reflected its early behavior. This early robot was capable of a wide range of complex behaviors including:

- Exploration: a combination of avoid-static-obstacle and noise.
- Hall following: move-ahead in the direction of the hall coupled with avoid-static-obstacle; enabled safe navigation down a corridor.
- Wall following or "drunken sailor" behavior: useful for going through doorways. A move-ahead schema pointing at an angle into the wall coupled with avoid-static-obstacle produced a behavior where the robot followed the wall and then passed through the first opening it found. This enabled the robot to complete nonspecific tasks such as "go down the hall and enter the first door on your right."
- Impatient waiting: occurred while the robot waited for a door to open. It consisted of a small amount of noise and a move-to-goal behavior targeted immediately beyond a closed door, coupled with the avoid-static-obstacle schema. The robot would oscillate in its local minima until the door opened. When the obstacle stimulus was no longer present because of the new opening, the robot moved through the doorway. This behavior is potentially useful for entering elevators, among other things. The behavior is also referred to as the fly-at-a-window behavior since a fly is attracted towards the light yet repelled by the glass at the window, and the behavior has a noisy component (panic?) that is actually helpful to the fly in finding openings that it may not initially sense.
- Indoor and outdoor navigation: demonstrated in several ways, including various combinations of the stay-on-path, avoid-static-obstacle, noise, move-ahead, and move-to-goal schemas.

- George: This Denning DRV-3, named after a fictitious Georgia Tech student, was the first robot to exhibit behavior-based docking (Arkin and Murphy 1990), teleautonomy (Arkin 1991), and avoid-past behaviors (Balch and Arkin 1993).
- Ren and Stimpy: A pair of Denning MRV-2 robots used for dodge, escape, forage, and multiagent behavioral research.
- Buzz: Used in the AAAI competition described in section 3.4.4 (Arkin et al. 1993).
- Io, Callisto, Ganymede: Three student-constructed small mobile robots for multiagent research and winners in a robot competition (Balch et al. 1995).
- Mobile manipulator: One of the MRV-2s fitted with a CRS+ robot arm (Cameron et al. 1993).

4.5 OTHER ARCHITECTURES

We now survey a representative sampling of the wide range of other behavior-based architectures that exist, highlighting each one's approach and contribution. All share a philosophy of commitment to sensing and action, elimination

Table 4.3
Circuit architecture

Name	Circuit Architecture
Background	Early reactive architecture
Precursors	Brooks 1986; Nilsson 1984; Barbera et al. 1984; Johnson 1983
Principal design method	Situated activity
Developers	L. Kaelbling and S. Rosenschein (SRI)
Response encoding	Discrete (rule based)
Coordination method	Hierarchical mediation (arbitration with abstraction)
Programming method	Rex and Gapps
Robots fielded	Flakey
References	Kaelbling 1986; Kaelbling and Rosenschein 1991

or reduction of symbolic representational knowledge, and the use of behavioral units as their primary building blocks. Each one's uniqueness arises from its choice of coordination mechanisms, the response encoding methods used, the behavioral granularity, and the design methodology employed.

4.5.1 Circuit Architecture

The circuit architecture is a hybridization of the principles of reactivity as typified by the subsumption architecture, the abstractions used in RCS (Barbera et al. 1984) and Shakey (Nilsson 1984), and the use of logical formalisms (Johnson 1983). Table 4.3 summarizes this approach. We discussed aspects of this architecture in section 3.2.4.2, in particular the role of logical formalisms and situated automata. One strength this approach provides involves the use of abstraction through the bundling of reactive behaviors into assemblages and by allowing arbitration to occur within each level of abstraction, that is, what the designers refer to as hierarchical mediation. Another advantage is the use of formal logic as a means for expressing the behaviors, permitting compilation into hardware and assisting with the verification of the performance of the resulting robotic system (Rosenschein and Kaelbling 1987).

The motivations for this architecture, according to its designers, are typical for behavior-based systems in general: *modularity*, permitting incremental development; *awareness*, tightly coupling sensing to action; and *robustness*, being able to perform despite unanticipated circumstances or sensor failure.

4.5.2 Action-Selection

Action-selection is an architectural approach developed by Pattie Maes in the late 1980s. It uses a dynamic mechanism for behavior selection. Instead of employing a predefined priority-based strategy typified by the subsumption approach, individual behaviors (competence modules) have associated activation levels that ultimately provide the basis for run-time arbitration. A competence module resembles a traditional AI robotic operator with preconditions, add lists and delete lists. Additionally, an activation level is associated with the module that ultimately governs its applicability at any particular time by being above some threshold. The activation level for any particular module is affected by the current situation, higher level goals, spreading activation due to previous or potentially succeeding events in time, or inhibition from conflicting modules. Activation levels also decay over time. The module with the highest activation level is chosen for execution from the set of all modules whose preconditions are satisfied. The selection process is repeated as rapidly as possible as the world's circumstances change about the agent.

Because there is no predefined layering of behaviors as in subsumption, it is harder in action-selection to predict the agent's global performance in a dynamic environment, and thus action-selection has a greater emergent quality. Several global parameters are used to tune the control system, all of which are related to the activation levels (e.g., activation threshold, amount of activation energy injected). An advantage of this strategy is flexibility and openness, as the system's responses are not hard-wired. As an agent's intentions can influence the activation parameters, higher level goals can also induce performance changes (see Norman-Shallice model from chapter 6). The action-selection approach also shares much in philosophy with schema theory (Arbib 1992), especially regarding the use of activation levels for controlling behavioral performance. Its primary perceived limitation is the lack of real implementations on actual robots and thus no evidence exists of how easily the current competence module formats would perform in real world robotic tasks. Table 4.4 summarizes this architecture's characteristics.

4.5.3 Colony Architecture

The colony architecture (Connell 1989b) is a direct descendent of the subsumption architecture that uses simpler coordination strategies (i.e., suppression only) and permits a more flexible specification of behavioral relations. The

Table 4.4
Action-selection architecture

Name	Action-selection
Background	Dynamic competition system
Precursors	Minsky 1986; Hillis 1988
Principal design method	Experimental
Developer	Pattie Maes (MIT)
Response encoding	Discrete
Coordination method	Arbitration via action-selection
Programming method	Competence modules
Robots fielded	Simulations only
References	Maes 1990; Maes 1989

Table 4.5
Colony architecture

Name	Colony architecture
Background	Descendent of subsumption architecture
Precursors	Minsky 1986; Brooks 1986
Principal design method	Ethologically guided/experimental
Developer	John Connell (IBM)
Response encoding	Discrete rule based
Coordination method	Priority-based arbitration with suppression only
Programming method	Similar to subsumption
Robots fielded	Herbert (mobile manipulator), wheelchair
References	Connell 1989; Connell 1989b

colony architecture permits a treelike ordering for behavioral priority as opposed to the total ordering of layered behaviors found in subsumption. A closer relationship to ethology was also developed, using models derived from animals such as the coastal snail to justify the use of pure suppression networks. The pinnacle of this architecture was Herbert, a robot designed to wander about the corridors of the MIT laboratory and retrieve soda cans for recycling using vision, ultrasound, and infrared proximity sensors. Table 4.5 summarizes the colony architecture's characteristics.

Table 4.6
Animate Agent Architecture

Name	Animate agent architecture
Background	RAP-based situated activity system
Precursors	Miller, Galanter, and Pribram 1960; Georgeff et al. 1986; Agre and Chapman 1987; Ullman 1985
Principal design method	Situated activity
Developer	R. James Firby (University of Chicago)
Response encoding	NAT based (continuous)
Coordination method	Sequencing
Programming method	RAP language
Robots fielded	Chip (trash-cleaning robot)
References	Firby 1989, 1995; Firby and Slack 1995

4.5.4 Animate Agent Architecture

The animate agent architecture adds two components to the RAPs discussed in section 3.1.3: a skills system and special-purpose problem solving modules (Firby 1995). The skills provide continuous environmental response, typically using a NAT encoding (Slack 1990) (section 3.3.2), whereas RAPs provide an assemblage mechanism for bundling skills useful in particular situations. In a sense, RAPs are related to FSA states and can be used to sequence through a collection of skills over time (Firby and Slack 1995). Situations, however, are used to define the states (or context) and provide the overall design basis. Table 4.6 summarizes this architecture's key points.

4.5.5 DAMN

The Distributed Architecture for Mobile Navigation (DAMN) boasts a rather provocative name. Developed by Rosenblatt (1995), initially at the Hughes AI Center and subsequently at Carnegie-Mellon University, this behavior-based system has a unique coordination mechanism. The behaviors in DAMN, which was initially touted as a fine-grained alternative to the subsumption architecture (Rosenblatt and Payton 1989), are themselves asynchronous processes each generating outputs as a collection of votes cast over a range of responses. The votes for each behavior can be cast in a variety of ways, including differing statistical distributions over a range of responses. The behavioral arbitration

Table 4.7
DAMN Architecture

Name	DAMN (Distributed Architecture for Mobile Navigation)
Background	Fine-grained subsumption-style architecture
Precursors	Brooks 1986; Zadeh 1973
Principal design method	Experimental
Developer	Julio Rosenblatt (CMU)
Response encoding	Discrete vote sets
Coordination method	multiple winner-take-all arbiters
Programming method	Custom
Robots fielded	DARPA ALV and UGV vehicles
References	Rosenblatt and Payton 1989; Rosenblatt 1995

method, discussed in section 3.4.3, is a winner-take-all strategy in which the response with the largest number of votes is selected for enactment. Table 4.7 highlights its characteristics.

Also unique to the DAMN architecture are the multiple parallel arbiters for both speed and turning control. Arbitration for each of these activities occurs completely independently. Chapter 9 revisits the DAMN architecture in examining the Defense Advanced Research Project Agency's (DARPA's) Unmanned Ground Vehicle Demo II Program.

4.5.6 Skill Network Architecture

The skill network architecture is a behavior-based system developed for graphical animation rather than robotics. Indeed, the use of behavior-based techniques within animation is becoming widespread. Pioneering work by Craig Reynolds (1987) on his Boids system provided a compelling set of visual behaviors for flocks of birds and schools of fish. Recent work by Hodgins and Brogan (1994) has extended these techniques to model not only graphical creatures' responses to their fictional environments but also their dynamics. Zeltzer and Johnson (1991), however, developed the skill network architecture in a more general way, providing for a variety of computing agents: sensing agents that provide information regarding the environment, skill agents that encode behaviors, and goal agents that monitor whether certain conditions have been met. A modification of Maes' action-selection mechanism serves as the basis

Table 4.8
Skill network architecture

Name	Skill network architecture
Background	Behavior-based animation architecture
Precursors	Maes 1989; Badler and Webber 1991
Principal design method	Ethological/Experimental
Developer	David Zeltzer (MIT)
Response encoding	Discrete
Coordination method	Action-selection
Programming method	Agent libraries
Robots fielded	Graphical animations only
References	Zeltzer and Johnson 1991

for coordination. Of particular note in this architecture is the designer's concern for mapping the multiple concurrent processes over a network of Unix workstations to minimize computation time, an essential aspect of computer-generated animation. The general characteristics of the skill network architecture appear in table 4.8.

4.5.7 Other Efforts

Several other behavioral approaches warrant mentioning.

- BART: The Behavioral Architecture for Robot Tasks, was an early approach (Kahn 1991) that defined task behaviors arbitrarily and provided support for military robotic missions. Of note was its use of a focus-of-attention manager to provide situational context for the selection of relevant behaviors. A BART language was developed for specifying behavioral tasks.
- Autochthonous behaviors: Developed by Grupen and Henderson (1990), this behavioral approach uses logical impedance controllers as the basis for specifying behavioral response. Of particular interest is its focus on grasping and manipulation tasks as opposed to robot navigation. Grupen and Henderson's method also relies far more heavily on representational knowledge (3D models generated by sensor data) than typical reactive systems.
- Anderson and Donath: A behavioral approach strongly influenced by ethological studies and fielded on Scarecrow, a 400-pound robot (Anderson and Donath 1991). The system, similar in spirit to the schema-based approach, uses

potential fields methods for the response encoding and vector summation for the coordination mechanism.

- SmartyCat: Defined in the SmartyCat Agent Language (Lim 1994), similar in flavor to the Behavior Language (Brooks 1990a), this behavior specification approach has been tested on a Cybermotion K2A robot.
- Dynamic Reaction: A behavior-based system capable of using goal-based constraints in dynamic or rapidly changing worlds (Sanborn 1988). This system was tested in simulation in trafficworld, a driving simulator.
- ARC (Artificial Reflex Control): In this model for robot control systems, actual biological reflexes serve as a basis for their design in robots. The model has particular relevance for rehabilitative robotics, in which a prosthetic device such as an artificial hand or limb must coordinate with an active human. This control model has been applied to hand, knee, and leg controllers for potential use within human-assistive technology (Bekey and Tomovic 1986, 1990).
- Niche Robot Architecture: This architecture, developed by Miller (1995), draws on the notion of ecological niches as espoused by MacFarland (McFarland and Bosser 1993) and presented in chapter 2. It focuses more on the philosophical issues of creating robots for specific tasks rather than on rigid commitments to specific behavioral encodings or coordination strategies. Several real world robots fit into this paradigm, including Rocky III, a prototype Mars microrover; Fuddbot, a simplistic vacuum cleaning robot; and a robotic wheelchair tasked with assisting the handicapped.

4.6 ARCHITECTURAL DESIGN ISSUES

We can extract a number of common threads from this diversity of architectural approaches as well as some themes driving the development of these systems.

- Analysis versus synthesis. This methodological difference relates to the underlying assumptions regarding just what intelligence is. In some instances, intelligence is perceived as something that can be reduced to an atomic unit that when appropriately organized and replicated can yield high-level intelligent action. In other approaches, abstract pieces of intelligent systems, often extracted from or motivated by biological counterparts, can be used to construct the required robotic performance.
- Top-down (knowledge-driven) versus bottom-up (data-driven) design. This aspect relates more closely to experimentation and discovery as a design driver versus a formal analysis and characterization of the requisite knowledge a system needs to possess to manifest intelligent robotic performance. These

differences perhaps parallel to a degree the "scruffy versus neat" dichotomy in AI.

- Domain relevance versus domain independence. To some extent this characteristic captures the view that there either is or is not a single form of intelligence. Here the AI parallel is "weak versus strong" methods.
- Understanding intelligence versus intelligent machines. The fundamental difference here lies in the designer's goals. Biological constraints can be applied in the development of a robotic architecture in an effort to understand the nature of animal intelligence. This approach may compromise the utility of the resulting machine intelligence. Often robot architects who follow this path have an underlying assumption that intelligence is fundamentally independent of the underlying substrate in which it is embedded. This is merely a working hypothesis, as there is yet no strong evidence to support it (nor to contradict it). Other architects are concerned with the more direct goal of building useful and productive machines with sufficient intelligence to function within the world in which they are situated. Whether these machines relate to biological systems is not their concern.

These competing goals and methods result in a wide range of architectural approaches, as we have seen throughout this chapter. Many roboticists feel that although behavior-based methods provide excellent responsiveness in dynamic environments, much is lost in their eliminating the use of representational knowledge. These researchers have considered how representational knowledge in various forms can be integrated into these behavioral architectures. In chapter 5, we will encounter various methods that can introduce representational knowledge into reactive robotic architectures while maintaining most, if not all, of their desirable properties. In chapter 6 we will study hybrid architectures that attempt to supplement behavior-based architectures with not only representational knowledge but additional deliberative planning capabilities. Chapter 8 discusses how learning and adaptation can be introduced into these systems, another very important research area.

4.7 CHAPTER SUMMARY

- A wide range of architectural solutions exist under the behavior-based paradigm.
- These architectures, in general, share an aversion to the use of representational knowledge, emphasis on a tight coupling between sensing and action, and decomposition into behavioral units.

- These architectures differ in the granularity of behavioral decomposition, coordination methods used, response encoding technique, basis for development, and other factors.
- Our working definition is that robotic architecture is the discipline devoted to the design of highly specific and individual robots from a collection of common software building blocks.
- Robotic architectures are similar in the sense that they are all Turing computable but indeed differ significantly in terms of their organizational components and structure.
- Behavior-based architectures can be evaluated in terms of their support for parallelism, hardware retargetability, ecological niche fitting, modularity support, robustness, flexibility, ease of development, and performance.
- The subsumption architecture is a layered architecture that uses arbitration strategies and augmented finite state machines as its basis. It has been implemented on many robotic systems using rule-based encodings and an experimental design methodology.
- Motor schemas are a software-oriented dynamic reactive architecture that is non-layered and cooperative (as opposed to competitive). Vectors serve as the continuous response encoding mechanism with summation as the fundamental coordination strategy. Several robotic systems have been implemented, and the architecture has had significant influence from biological considerations.
- Circuit architectures predominantly use logical expressions for behavioral encoding, use abstraction coupled with arbitration, and typically follow the situated activity design paradigm.
- Action-selection architectures are dynamic rather than fixed competition systems, and they also use arbitration.
- The colony architecture is a simplified version of subsumption, more straightforward in its implementation.
- The animate agent architecture uses reactive action packages (RAPs) and sequencing methods to unfold situational responses over time.
- The DAMN architecture provides voting mechanisms for behavioral response encodings, with a winner-take-all arbitration mechanism in the style of subsumption.
- The skill network architecture is particularly well suited for graphical animation and uses action-selection techniques.
- Many other behavior-based architectures also exist, varying at some level from the other architectural systems.

- Design choices for robotic architects involve issues such as whether to use analysis or synthesis, take a top-down or bottom-up design stance, design for specific domains or be more general, and whether to consider the abstract role of intelligence in general or simply be concerned with building smarter machines.