Introduction to AI Robotics

Robin R. Murphy

A Bradford Book
The MIT Press
Cambridge, Massachusetts
London, England

Contents

What are Robotic Paradigms?

PARADIGM

A *paradigm is a philosophy or set of assumptions and/or techniques which characterize an approach to a class of problems*. It is both a way of looking at the world and an implied set of tools for solving problems. No one paradigm is right; rather, some problems seem better suited for different approaches. For example, consider calculus problems. There are problems that could be solved by differentiating in cartesian (X, Y, Z) coordinates, but are much easier to solve if polar coordinates (r, θ) are used. In the domain of calculus problems, Cartesian and polar coordinates represent two different paradigms for viewing and manipulating a problem. Both produce the correct answer, but one takes less work for certain problems.

Applying the right paradigm makes problem solving easier. Therefore, knowing the paradigms of AI robotics is one key to being able to successfully program a robot for a particular application. It is also interesting from a historical perspective to work through the different paradigms, and to examine the issues that spawned the shift from one paradigm to another.

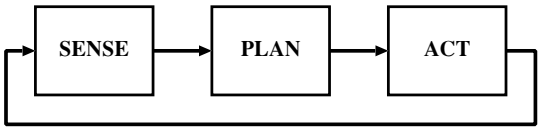
ROBOTIC PARADIGMS

There are currently three paradigms for organizing intelligence in robots: hierarchical, reactive, and hybrid deliberative/reactive. The paradigms are described in two ways:

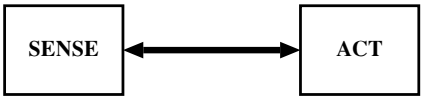
ROBOTIC PARADIGM PRIMITIVES

1. **By the relationship between the three commonly accepted primitives of robotics: SENSE, PLAN, ACT.** The functions of a robot can be divided into three very general categories. If a function is taking in information from the robot's sensors and producing an output useful by other functions, then that function falls in the SENSE category. If the function is taking in information (either from sensors or its own knowledge about how the world works) and producing one or more tasks for the robot to perform (go down the hall, turn left, proceed 3 meters and stop), that function is in the PLAN category. Functions which produce output commands to motor actuators fall into ACT (turn $98°$, clockwise, with a turning velocity of 0.2mps). Fig. I.2 attempts to define these three primitives in terms of inputs and outputs; this figure will appear throughout the chapters in Part I.

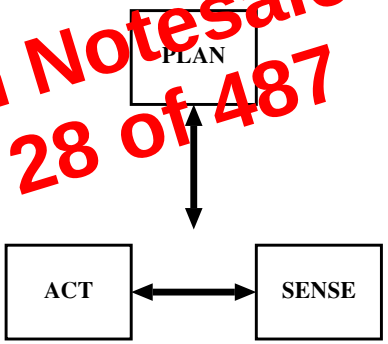
2. **By the way sensory data is processed and distributed through the system**. How much a person or robot or animal is influenced by what it senses. So it is often difficult to adequately describe a paradigm with just a box labeled SENSE. In some paradigms, sensor information is restricted to being used in a specific, or dedicated, way for each function of a robot;



a.



b.



c.

Figure I.3 Three paradigms: a.) Hierarchical, b.) Reactive, and c.) Hybrid deliberative/reactive.

turns out to be very hard and brittle due to the *frame problem* and the need for a *closed world assumption*.

Fig. I.4 shows how the Hierarchical Paradigm can be thought of as a transitive, or Z-like, flow of events through the primitives given in Fig. I.4. Unfortunately, the flow of events ignored biological evidence that sensed information can be directly coupled to an action, which is why the sensed information input is blacked out.

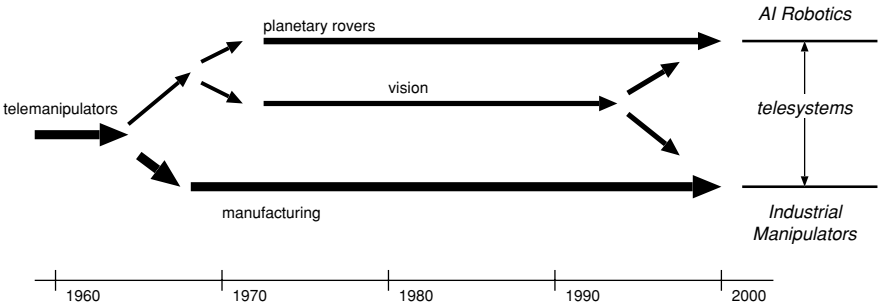


Figure 1.1 A timeline showing forks in development of robots.

placed on the mechanical aspects of the robot to ensure precision and repeatability and methods to make sure the robot could move precisely and repeatable, quickly enough to make a profit. Because assembly lines were engineered to mass produce a certain product, the robot didn't have to be able to notice any problems. The standards for mass production would make it more economical to devise mechanisms that would ensure parts would be in the correct place. A robot for automation could essentially be blind and senseless.

Robotics for the space program took a different fork, concentrating instead on highly specialized, one-of-a-kind planetary rovers. Unlike a highly automated manufacturing plant, a planetary rover operating on the dark side of the moon (no radio communication) might run into unexpected situations. Consider that on Apollo 17, astronaut and geologist Harrison Schmitt found an orange rock on the moon; an orange rock was totally unexpected. Ideally, a robot would be able to notice something unusual, stop what it was doing (as long as it didn't endanger itself) and investigate. Since it couldn't be preprogrammed to handle all possible contingencies, it had to be able to notice its environment and handle any problems that might occur. At a minimum, a planetary rover had to have some source of sensory inputs, some way of interpreting those inputs, and a way of modifying its actions to respond to a changing world. And the need to sense and adapt to a partially unknown environment is the need for intelligence.

The fork toward AI robots has not reached a termination point of truly autonomous, intelligent robots. In fact, as will be seen in Ch. 2 and 4, it wasn't until the late 1980's that any visible progress toward that end was made. So what happened when someone had an application for a robot which needed

try; processing immune suppressant drugs may expose workers to highly toxic chemicals.

Another example of a task that poses significant risk to a human is space exploration. People can be protected in space from the hard vacuum, solar radiation, etc., but only at great economic expense. Furthermore, space suits are so bulky that they severely limit an astronaut's ability to perform simple tasks, such as unscrewing and removing an electronics panel on a satellite. Worse yet, having people in space necessitates more people in space. Solar radiation embrittlement of metals suggests that astronauts building a large space station would have to spend as much time repairing previously built portions as adding new components. Even more people would have to be sent into space, requiring a larger structure. the problem escalates. A study by Dr. Jon Erickson's research group at NASA Johnson Space Center argued that a manned mission to Mars was not feasible without robot drones capable of constantly working outside of the vehicle to repair problems introduced by deadly solar radiation.[51] (Interestingly enough, a team of three robots which did just this were featured in the 1972 film, *Silent Running*, as well as by a young R2D2 in *The Phantom Menace*.)

Nuclear physics and space exploration are activities which are often far removed from everyday life, and applications where robots figure more prominently in the future than in current times.

The most obvious use of robots is manufacturing, where repetitious activities in unpleasant surroundings make human workers inefficient or expensive to retain. For example, robot "arms" have been used for welding cars on assembly lines. One reason that welding is now largely robotic is that it is an unpleasant job for a human (hot, sweaty, tedious work) with a low tolerance for inaccuracy. Other applications for robots share similar motivation: to automate menial, unpleasant tasks—usually in the service industry. One such activity is janitorial work, especially maintaining public rest rooms, which has a high turnover in personnel regardless of payscale. The janitorial problem is so severe in some areas of the US, that the Postal Service offered contracts to companies to research and develop robots capable of autonomously cleaning a bathroom (the bathroom could be designed to accommodate a robot).

Agriculture is another area where robots have been explored as an economical alternative to hard to get menial labor. Utah State University has been working with automated harvesters, using GPS (global positioning satellite system) to traverse the field while adapting the speed of harvesting to the rate of food being picked, much like a well-adapted insect. The De-

olence in 1812, legislation was passed to end worker violence and protect the mills. The rebelling workers were persecuted. While the Luddite movement may have been motivated by a quality-of-life debate, the term is often applied to anyone who objects to technology, or "progress," for any reason. The connotation is that Luddites have an irrational fear of technological progress.

The impact of robots is unclear, both what is the real story and how people interact with robots. The HelpMate Robotics, Inc. robots and janitorial robots appear to be competing with humans, but are filling a niche where it is hard to get human workers at any price. Cleaning office buildings is menial and boring, plus the hours are bad. One janitorial company has now invested in mobile robots through a Denver-based company, Continental Divide Robotics, citing a 90% yearly turnover in staff, even with profit sharing after two years. The Robotics Industries Association, a trade group, produces annual reports outlining the need for robotics, yet possibly the biggest robot money makers are in the entertainment and toy industries.

The cultural implications of robotics cannot be ignored. While the sheep shearing robots in Australia were successful and were ready to be commercialized for significant economic gains, the sheep industry reportedly rejected the robots. One story goes that the sheep ranchers would not accept a robot shearer unless it had a 0% fatality rate (it's apparently fairly easy to nick an artery on a squirming sheep). But human shearers accidently kill several sheep, while the robots had a demonstrably better rate. The use of machines raises an ethical question: is it acceptable for an animal to die at the hands of a machine rather than a person? What if a robot was performing a piece of intricate surgery on a human?

1.4 A Brief History of Robotics

Robotics has its roots in a variety of sources, including the way machines are controlled and the need to perform tasks that put human workers at risk.

In 1942, the United States embarked on a top secret project, called the Manhattan Project, to build a nuclear bomb. The theory for the nuclear bomb had existed for a number of years in academic circles. Many military leaders of both sides of World War II believed the winner would be the side who could build the first nuclear device: the Allied Powers led by USA or the Axis, led by Nazi Germany.

One of the first problems that the scientists and engineers encountered was handling and processing radioactive materials, including uranium and

Exercise 1.5

What is a Luddite?

Exercise 1.6

Describe at least two differences between AI and Engineering approaches to robotics.

Exercise 1.7

List three problems with teleoperation.

Exercise 1.8

Describe the components and the responsibilities of the local and the remote members of a telesystem.

Exercise 1.9

Describe the difference between telepresence and semi-autonomous control.

Exercise 1.10

List the six characteristics of applications that are well suited for teleoperation. Give at least two examples of potentially good applications for teleoperation not covered in the chapter.

Exercise 1.11 [*World Wide Web*]

Search the world wide web for sites that permit clients to use a robot remotely (one example is Xavier at Carnegie Mellon University). Decide whether each site is using human supervisory or shared control, and justify your answer.

Exercise 1.12 [*World Wide Web*]

Search the world wide web for applications and manufacturers of intelligent robots.

Exercise 1.13 [*World Wide Web*]

Dr. Harrison "Jack" Schmitt is a vocal proponent for space mining of Near Earth Objects (NEOs) such as mineral-rich asteroids. Because of the economics of manned mission, the small size of NEOs, human safety concerns, and the challenges of working in micro-gravity, space mining is expected to require intelligent robots. Search the web for more information on space mining, and give examples of why robots are needed.

Exercise 1.14 [*Programming*]

(This requires a robot with an on-board video camera and a teleoperation interface.) Teleoperate the robot through a slalom course of obstacles while keeping the robot in view as if controlling a RC car. Now looking only at the output of the video camera, repeat the obstacle course. Repeat the comparison several times, and keep track of the time to complete the course and number of collisions with obstacles. Which viewpoint led to faster completion of the course? Fewer collisions? Why?

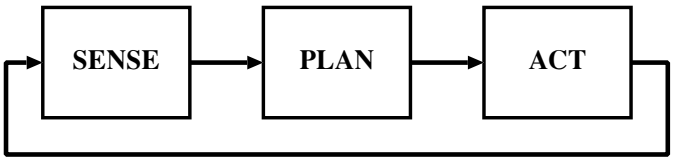


Figure 2.2 S,P,A organization of Hierarchical Paradigm.

ROBOT PRIMITIVES	INPUT	OUTPUT
SENSE	Sensor data →	Sensed information
PLAN	Information (sensed and/or cognitive) →	Directives
ACT	[redacted] directives	→ Actuator commands

Figure 2.3 Alternative description of how the 3 primitives interact in the Hierarchical Paradigm.

goal. Finally, the robot must carry out the first directive. After the robot has carried out the SENSE-PLAN-ACT sequence, it begins the cycle again: eyes open, the robot senses the consequence of its action, replans the directives (even though the directives may not have changed), and acts.

As shown in Fig. 2.3, sensing in the Hierarchical Paradigm is monolithic: all the sensor observations are fused into one global data structure, which the planner accesses. The global data structure is generally referred to as a *world model*. The term world model is very broad; "world" means both the outside world, and whatever meaning the robot ascribes to it. In the Hierarchical Paradigm, the world model typically contains

WORLD MODEL

A PRIORI

1. an *a priori* (previously acquired) representation of the environment the robot is operating in (e.g., a map of the building),
2. sensing information (e.g., "I am in a hallway, based on where I've traveled, I must be in the northwest hallway"), plus
3. any additional cognitive knowledge that might be needed to accomplish a task (e.g., all packages received in the mail need to be delivered to Room 118).

ing the 1970's and 1980's worked on either computer vision related issues, trying to get the robots to be able to better sense the world, or on path planning, computing the most efficient route around obstacles, etc. to a goal location.

2.4 Representative Architectures

As mentioned in Part I an architecture is a method of implementing a paradigm, of embodying the principles in some concrete way. Ideally, an architecture is generic; like a good object-oriented program design, it should have many reusable pieces for other robot platforms and tasks.

Possibly the two best known architectures of the Hierarchical period are the Nested Hierarchical Controller (NHC) developed by Meystel[93] and the NIST Realtime Control System (RCS) originally developed by Albus[1] with its teleoperation version for JPL called NASREM.

2.4.1 Nested Hierarchical Controller

As shown in Fig. 2.6, the Nested Hierarchical Controller architecture has components that are easily identified as either SENSE, PLAN, or ACT. The robot begins by gathering observations from its sensors and combining those observations to form the World Model data structure through the SENSE activity. The World Model may also contain *a priori* knowledge about the world, for example, maps of a building, rules saying to stay away from the foyer during the start and finish of business hours, etc. After the World Model has been created or updated, then the robot can PLAN what actions it should take. Planning for navigation has a local procedure consisting of three steps executed by the Mission Planner, Navigator, and Pilot. Each of these modules has access to the World Model in order to compute their portion of planning. The last step in planning is for the Pilot module to generate specific actions for the robot to do (e.g., Turn left, turn right, move straight at a velocity of 0.6 meters per second). These actions are translated into actuator control signals (e.g., Velocity profile for a smooth turn) by the Low-Level Controller. Together, the Low-Level Controller and actuators form the ACT portion of the architecture.

The major contribution of NHC was its clever decomposition of planning into 3 different functions or subsystems aimed at supporting navigation: the *Mission Planner*, the *Navigator*, and the *Pilot*. As shown in Fig. 2.6, the Mission Planner either receives a mission from a human or generates a mission

MISSION PLANNER
NAVIGATOR
PILOT

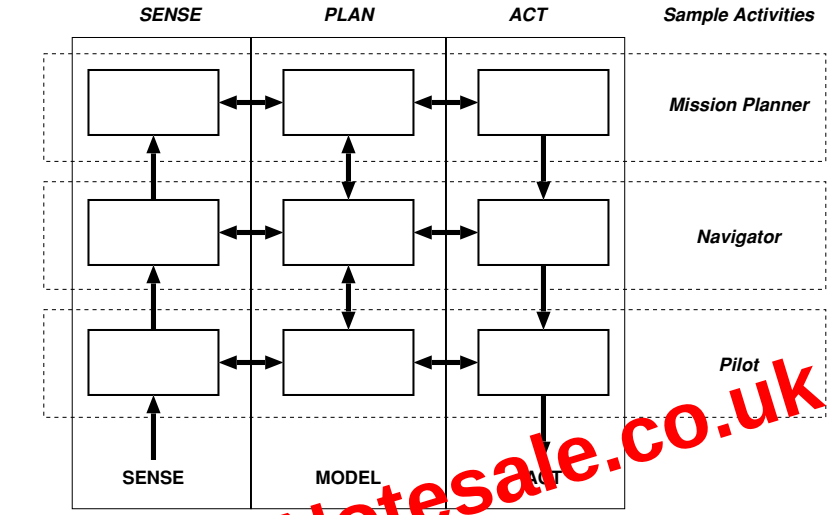
NHC has several advantages. It differs from Strips in that it interleaves planning and acting. The robot comes up with a plan, starts executing it, then changes it if the world is different than it expected. Notice that the decomposition is inherently hierarchical in intelligence and scope. The Mission Planner is "smarter" than the Navigator, who is smarter than the Pilot. The Mission Planner is responsible for a higher level of abstraction then the Navigator, etc. We will see that other architectures, both in the Hierarchical and Hybrid paradigms, will make use of the NHC organization.

One disadvantage of the NHC decomposition of the planning function is that it is appropriate only for navigation tasks. The division of responsibilities seems less helpful, or clear, for tasks such as picking up a box, rather than just moving over to it. The role of a Pilot in controlling end-effectors is not clear. At the time of its initial development, NHC was never implemented and tested on a real mobile robot; hardware costs during the Hierarchical period forced most roboticists to work in simulation.

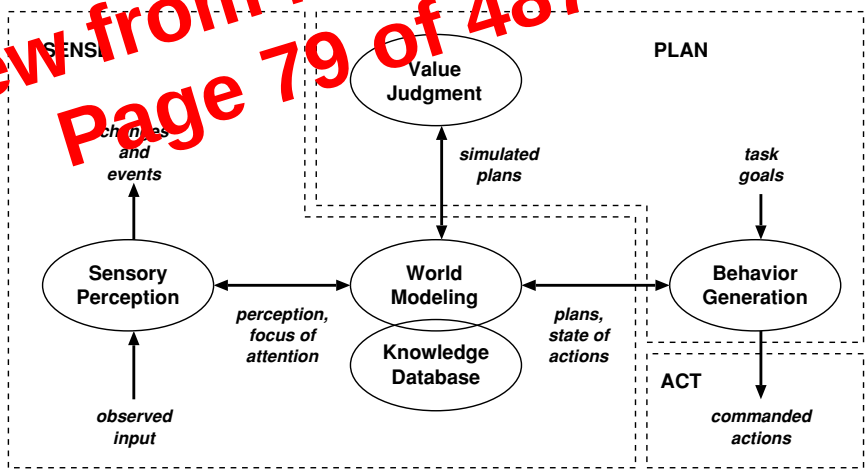
2.4.2 NIST RCS

Jim Albus at the National Bureau of Standards (later renamed the National Institute of Standards and Technology or NIST) anticipated the need for intelligent industrial manipulators, even as engineering and AI researchers were splitting into their groups. He saw that one of the major obstacles in applying AI to manufacturing robots was that there were no common terms, no common set of design standards. This made industry and equipment manufacturers leery of AI, for fear of buying an expensive robot that would not be compatible with robots purchased in the future. He developed a very detailed architecture called the Real-time Control System (RCS) Architecture to serve as a guide for manufacturers who wanted to add more intelligence to their robots. RCS used NHC in its design, as shown in Fig. 2.7.

SENSE activities are grouped into a set of modules under the heading of sensory perception. The output of the sensors is passed off to the world modeling module which constructs a global map using information in its associated knowledge database about the sensors and any domain knowledge (e.g., the robot is operating underwater). This organization is similar to NHC. The main difference is that the sensory perception module introduces a useful preprocessing step between the sensor and the fusion into a world model. As will be seen in Ch. 6, sensor preprocessing is often referred to as *feature extraction*.



a.



b.

Figure 2.7 Layout of RCS: a.) hierarchical layering of sense-model-act, and b.) functional decomposition.

use of the word "reactive" in ethology is at odds with the way the word is used in robotics. In ethology, reactive behavior means learned behaviors or a skill; in robotics, it connotes a reflexive behavior. If the reader is unaware of these differences, it may be hard to read either the ethological or AI literature without being confused.

3.2.1 Reflexive behaviors

Reflexive types of behaviors are particularly interesting, since they imply no need for any type of cognition: if you sense it, you do it. For a robot, this would be a hardwired response, eliminating computation and guaranteed to be fast. Indeed, many kit or hobby robots work off of reflexes, represented by circuits.

Reflexive behaviors can be further divided into three categories:[1]

REFLEXES 1. *reflexes*: where the response lasts only as long as the stimulus, and the response is proportional to the intensity of the stimulus.

TAXES 2. *taxes*: where the response is to move to a particular orientation. Baby turtles exhibit *tropotaxis*; they are hatched at night and move to the brightest light. Until recently the brightest light would be the ocean reflecting the moon, but the intrusion of man has changed that. Owners of beach front property in Florida now have to turn off their outdoor lights during hatching season to avoid the lights being a source for tropotaxis. Baby turtles hatch at night, hidden from shore birds who normally eat them. It had been a mystery as to how baby turtles knew which way was the ocean when they hatched. The story goes that a volunteer left a flashlight on the sand while setting up an experiment intended to show that the baby turtles used magnetic fields to orient themselves. The magnetic field theory was abandoned after the volunteers noticed the baby turtles heading for the flashlight! Ants exhibit a particular taxis known as *chemotaxis*; they follow trails of pheromones.

FIXED-ACTION PATTERNS 3. *fixed-action patterns*: where the response continues for a longer duration than the stimulus. This is helpful for fleeing predators. It is important to keep in mind that a taxis can be any orientation relative to a stimulus, not just moving towards.

The above categories are not mutually exclusive. For example, an animal going over rocks or through a forest with trees to block its view might persist

(fixed-action patterns) in orienting itself to the last sensed location of a food source (taxis) when it loses sight of it.

The tight coupling of action and perception can often be quantified by mathematical expressions. An example of this is orienting in angelfish. In order to swim upright, an angelfish uses an internal (*idiothetic*) sense of gravity combined with its vision sense (*allothetic*) to see the external percept of the horizon line of the water to swim upright. If the fish is put in a tank with prisms that make the horizon line appear at an angle, the angelfish will swim cockeyed. On closer inspection, the angle that the angelfish swims at is the vector sum of the vector parallel to gravity with the vector perpendicular to the perceived horizon line! The ability to quantify animal behavior suggests that computer programs can be written which do likewise.

IDIOTHETIC
ALLOTHETIC

3.3 Coordination and Control of Behaviors

KONRAD LORENZ
NIKO TINBERGEN

Konrad Lorenz and Niko Tinbergen were the founding fathers of ethology. Each man independently became fascinated not only with individual behaviors of animals, but how animals acquired behaviors and selected or coordinated sets of behaviors. Their work provided some insight into four different ways an animal might acquire and organize behaviors. Lorenz and Tinbergen's work also helps with the computational theory Level 2 understanding of how to make a process out of behaviors.

The four ways to acquire a behavior are:

INNATE

1. to be born with a behavior (*innate*). An example is the feeding behavior in baby arctic terns. Arctic terns, as the name implies, live in the Arctic where the terrain is largely shades of black and white. However, the Arctic tern has a bright reddish beak. When babies are hatched and are hungry, they peck at the beak of their parents. The pecking triggers a regurgitation reflex in the parent, who literally coughs up food for the babies to eat. It turns out that the babies do not recognize their parents, per se. Instead, they are born with a behavior that says: if hungry, peck at the largest red blob you see. Notice that the only red blobs in the field of vision should be the beaks of adult Arctic terns. The largest blob should be the nearest parent (the closer objects are, the bigger they appear). This is a simple, effective, and computationally inexpensive strategy.

SEQUENCE OF INNATE BEHAVIORS

2. to be born with a *sequence of innate behaviors*. The animal is born with a sequence of behaviors. An example is the mating cycle in digger wasps.

with predator movement) or a group of neurons which do the equivalent of a computer algorithm.

COMPOUND RELEASERS

Another important point about IRMs is that the releaser can be a *compound of releasers*. Furthermore, the releaser can be a combination of either external (from the environment) or internal (motivation). If the releaser in the compound isn't satisfied, the behavior isn't triggered. The pseudo-code below shows a compound releaser.

```
enum            Releaser={PRESENT, NOT_PRESENT};
Releaser        food;
while (TRUE)
{
   food = senseFood();
   hungry = checkState();
      if (food == PRESENT && hungry==PRESENT)
      feed();
}
```

IMPLICIT CHAINING

The next example below shows what happens in a sequence of behaviors, where the agent eats, then nurses its young, then sleeps, and repeats the sequence. The behaviors are implicitly chained together by their releasers. Once the initial releaser is encountered, the first behavior occurs. It executes for one second (one "movement" interval), then control passes to the next statement. If the behavior isn't finished, the releasers remain unchanged and no other behavior is triggered. The program then loops to the top and the original behavior executes again. When the original behavior has completed, the internal state of the animal may have changed or the state of the environment may have been changed as a result of the action. When the motivation and environment match the stimulus for the releaser, the second behavior is triggered, and so on.


```
enum            Releaser={PRESENT, NOT_PRESENT};
Releaser        food, hungry, nursed;
while (TRUE) {
   food = sense();
   hungry = checkStateHunger();
   child = checkStateChild();
   if (hungry==PRESENT)
      searchForFood(); //sets food = PRESENT when done
   if (hungry==PRESENT && food==PRESENT)
      feed(); // sets hungry = NOT_PRESENT when done
```

Gibson referred to his work as an "ecological approach" because he believed that perception evolved to support actions, and that it is silly to try to discuss perception independently of an agent's environment, and its survival behaviors. For example, a certain species of bees prefers one special type of poppy. But for a long time, the scientists couldn't figure out how the bees recognized that type of poppy because as color goes, it was indistinguishable from another type of poppy that grows in the same area. Smell? Magnetism? Neither. They looked at the poppy under UV and IR light. In the non-visible bands that type of poppy stood out from other poppy species. And indeed, the scientists were able to locate retinal components sensitive to that bandwidth. The bee and poppy had co-evolved, where the poppy's color evolved to a unique bandwidth while at the same time the bee's retina was becoming specialized at detecting that color. With a retina "tuned" for the poppy, the bee didn't have to do any reasoning about whether there was a poppy in view, and, if so, was it the right species of poppy. If the color was present, the poppy was there.

Fishermen have exploited affordances since the beginning of time. A fishing lure attempts to emphasize those aspects of a fish's desired food, presenting the strongest stimulus possible. If the fish is hungry, the stimulus of the lure will trigger feeding. As seen in Fig. 3.6, fishing lures often look to a human almost nothing like the bait they imitate.

DIRECT PERCEPTION

What makes Gibson's interesting to roboticists is that an affordance is directly perceivable. Direct perception means that the sensing process doesn't require memory, inference, or interpretation. This means minimal computation, which usually translates to very rapid execution times (near instantaneous) on a computer or robot.

But can an agent actually perceive anything meaningful without some memory, inference, or interpretation? Well, certainly baby arctic terns don't need memory or inference to get food from a parent. And they're definitely not interpreting red in the sense of: "oh, there's a red blob. It's a small oval, which is the right shape for Mom, but that other one is a square, so it must be a graduate ethology student trying to trick me." For baby arctic terns, it's simply: `red = food, bigger red = better`.

Does this work for humans? Consider walking down the hall and somebody throws something at you. You will most likely duck. You also probably ducked without recognizing the object, although later you may determine it was only a foam ball. The response happens too fast for any reasoning: "Oh look, something is moving towards me. It must be a ball. Balls are usually hard. I should duck." Instead, you probably used a phenomena so basic that

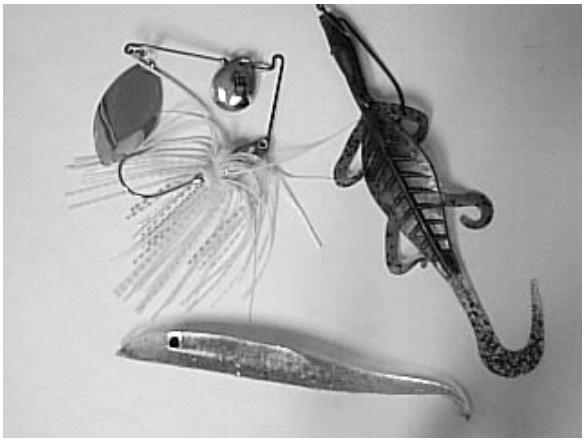


Figure 3.6 A collection of artificial bait, possibly the first example of humans exploiting affordances. Notice that the lures exaggerate one or more attributes of what a fish might eat.

OPTIC FLOW

you haven't noticed it, called *optic flow*. Optic flow is a neural mechanism for determining motion. Animals can determine time to contact quite easily with it. You probably are somewhat familiar with optic flow from driving in a car. When driving or riding in a car, objects in front seem to be in clear focus but the side of the road is a little blurry from the speed. The point in space that the car is moving to is the focus of expansion. From that point outward, there is a blurring effect. The more blurring on the sides, the faster the car is going. (They use this all the time in science fiction movies to simulate faster-than-light travel.) That pattern of blurring is known as a flow field (because it can be represented by vectors, like a gravitational or magnetic field). It is straightforward, neurally, to extract the time to contact, represented in the cognitive literature by τ.

TIME TO CONTACT

Gannets and pole vaulters both use optic flow to make last-minute, precise movements as reflexes. Gannets are large birds which dive from high altitudes after fish. Because the birds dive from hundreds of feet up in the air, they have to use their wings as control surfaces to direct their dive at the targeted fish. But they are plummeting so fast that if they hit the water with their wings open, the hollow bones will shatter. Gannets fold their wings just before hitting the water. Optic flow allows the time to contact, τ, to be a stimulus: when the time to contact dwindles below a threshold, fold those wings!

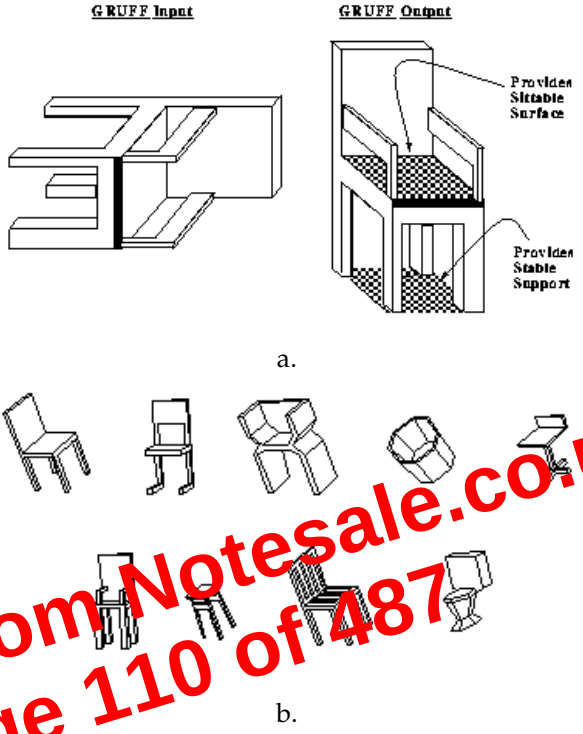


Figure 3.7 The GRUFF system: a.) input, and b.) different types of chairs recognized by GRUFF. (Figures courtesy of Louise Stark.)

the function of sittability. And that affordance of sittability should be something that can be extracted from an image:

- Without memory (the agent doesn't need to memorize all the chairs in the world).
- Without inference (the robot doesn't need to reason: "if it has 4 legs, and a seat and a back, then it's a chair; we're in an area which should have lots of chairs, so this makes it more likely it's a chair").
- Without an interpretation of the image (the robot doesn't need to reason: "there's an arm rest, and a cushion, ..."). A computer should just be able to look at a picture and say if something in that picture is sittable or not.

to use for different environmental conditions. Schema theory is expressive enough to represent basic concepts like IRMs, plus it supports building new behaviors out of primitive components. This will be discussed in more detail in later chapters.

This alternative way of creating a behavior by choosing between alternative perceptual and motor schemas can be thought of as:

Behavior::Schema

Data	environmental_state
Methods	choose_PS(environmental_state) perceptual_schema_1() perceptual_schema_2() motor_schema()

RANA COMPUTATRIX

Arbib and colleagues did work constructing computer models of visually guided behaviors in frogs and toads. They used schema theory to represent the toad's behavior in computational terms, and called their model *rana computatrix* (rana is the classification for toads and frogs). The model explained Ingle's observations as to what occasionally happens when a toad sees two flies at once.[3] Toads and frogs can be characterized as responding visually to either small, moving objects and large, moving objects. Small, moving objects release the feeding behavior, where the toad orients itself towards the object (taxis) and then snaps at it. (If the object turns out not to be a fly, the toad can spit it out.) Large moving objects release the fleeing behavior, causing the toad to hop away. The feeding behavior can be modeled as a behavioral schema, or template, shown in Fig. 3.9.

When the toad sees a fly, an instance of the behavior is instantiated; the toad turns toward that object and snaps at it. Arbib's group went one level further on the computational theory.[7] They implemented the taxis behavior as a vector field: rana computatrix would literally feel an attractive force along the direction of the fly. This direction and intensity (magnitude) was represented as a vector. The direction indicated where rana had to turn and the magnitude indicated the strength of snapping. This is shown in Fig. 3.10.

What is particularly interesting is that the rana computatrix program predicts what Ingle saw in real toads and frogs when they are presented with two flies simultaneously. In this case, each fly releases a separate instance of the feeding behavior. Each behavior produces the vector that the toad needs to turn to in order to snap at that fly, without knowing that the other be-

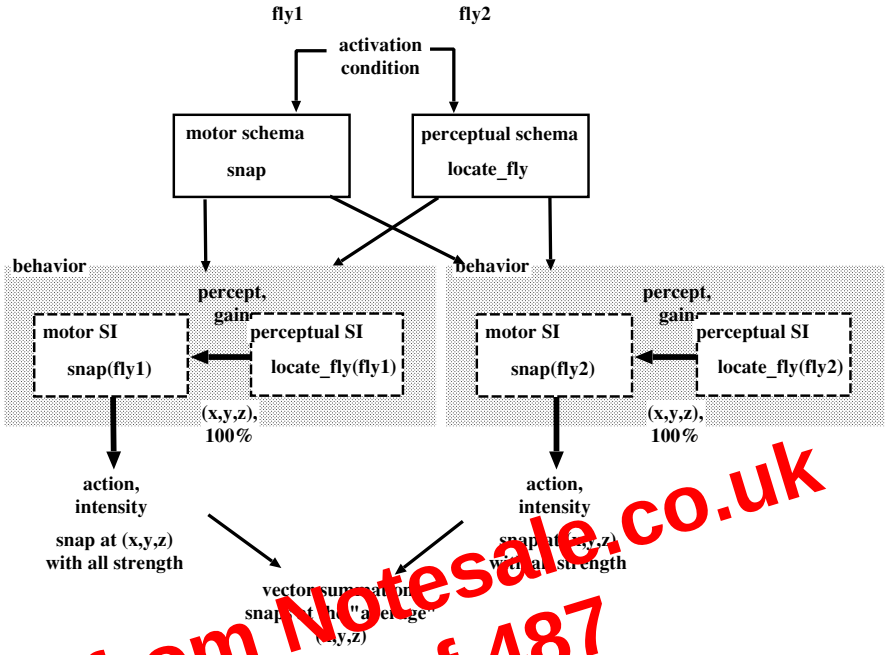


Figure 3.11 Schema theory of a frog snapping at a fly when presented with two flies equidistant.

it also inhibits the perceptual schema for feeding. As a result, the inhibition keeps the frog from trying to both flee from predators and eat them.

3.6 Principles and Issues in Transferring Insights to Robots

PRINCIPLES FOR PROGRAMMING

To summarize, some general principles of natural intelligence which may be useful in programming robots:

- Programs should decompose complex actions into independent behaviors, which tightly couple sensing and acting. Behaviors are inherently parallel and distributed.

- In order to simplify control and coordination of behaviors, an agent should rely on straightforward, boolean activation mechanisms (e.g. IRM).

- In order to simplify sensing, perception should filter sensing and consider only what is relevant to the behavior (action-oriented perception).
- Direct perception (affordances) reduces the computational complexity of sensing, and permits actions to occur without memory, inference, or interpretation.
- Behaviors are independent, but the output from one 1) may be combined with another to produce a resultant output, or 2) may serve to inhibit another (competing-cooperating).

Unfortunately, studying natural intelligence does not give a complete picture of how intelligence works. In particular there are several unresolved issues:

UNRESOLVED ISSUES

- *How to resolve conflicts between concurrent behaviors?* Robots will be required to perform concurrent tasks; for example, a rescue robot sent in to evacuate a building will have to navigate hallways while looking for rooms to examine for people, as well as look for signs of a spreading fire. Should the designer specify dominant behaviors? Combine? Let conflicting behaviors cancel and have alternative behavior triggered? Indeed, one of the biggest divisions in robot architectures is how they handle concurrent behaviors.
- *When are explicit knowledge representations and memory necessary?* Direct perception is wonderful in theory, but can a designer be sure that an affordance has not been missed?
- *How to set up and/or learn new sequences of behaviors?* Learning appears to be a fundamental component of generating complex behaviors in advanced animals. However, the ethological and cognitive literature is unsure of the mechanisms for learning.

It is also important to remember that natural intelligence does not map perfectly onto the needs and realities of programming robots. One major advantage that animal intelligence has over robotic intelligence is evolution. Animals evolved in a way that leads to survival of the species. But robots are expensive and only a small number are built at any given time. Therefore, individual robots must "survive," not species. This puts tremendous pressure on robot designers to get a design right the first time. The lack of evolutionary pressures over long periods of time makes robots extremely vulnerable to design errors introduced by a poor understanding of the robot's ecology.

Exercise 3.2

Explain in one or two sentences each of the following terms: reflexes, taxes, fixed-action patterns, schema, affordance.

Exercise 3.3

Represent a schema, behavior, perceptual schema, and motor schema with an Object-Oriented Design class diagram.

Exercise 3.4

Many mammals exhibit a camouflage meta-behavior. The animal freezes when it sees motion (an affordance for a predator) in an attempt to become invisible. It persists until the predator is very close, then the animal flees. (This explains why squirrels freeze in front of cars, then suddenly dash away, apparently flinging themselves under the wheels of a car.) Write pseudo-code of the behaviors involved in the camouflage behavior in terms of innate releasing mechanisms, identifying the releasers for each behavior.

Exercise 3.5

Consider a mosquito hunting for a warm-blooded mammal and a good place to bite them. Identify the affordance for a warm-blooded mammal and the associated behavior. Represent this with schema theory (perceptual and motor schemas).

Exercise 3.6

One method for representing an IRM topic is to use finite state automata (FSA), which are commonly used in computer science. If you have seen FSAs, consider a FSA where the behaviors are states and releasers serve as the transitions between states. Express the sequence of behaviors in a female digger wasp as a FSA.

Exercise 3.7

Lego Mindstorms and Rug Warrior kits contain sensors and actuators which are coupled together in reflexive behaviors. Build robots which:

a. Reflexive avoid: turn left when they touch something (use touch sensor and two motors)

b. Phototaxis: follow a black line (use the IR sensor to detect the difference between light and dark)

c. Fixed-action pattern avoid: back up and turn right when robot encounters a "negative obstacle" (a cliff)

Exercise 3.8

What is the difference between direct perception and recognition?

Exercise 3.9

Consider a cockroach, which typically hides when the lights are turned on. Do you think the cockroach is using direct perception or recognition of a hiding place? Explain why. What are the percepts for the cockroach?

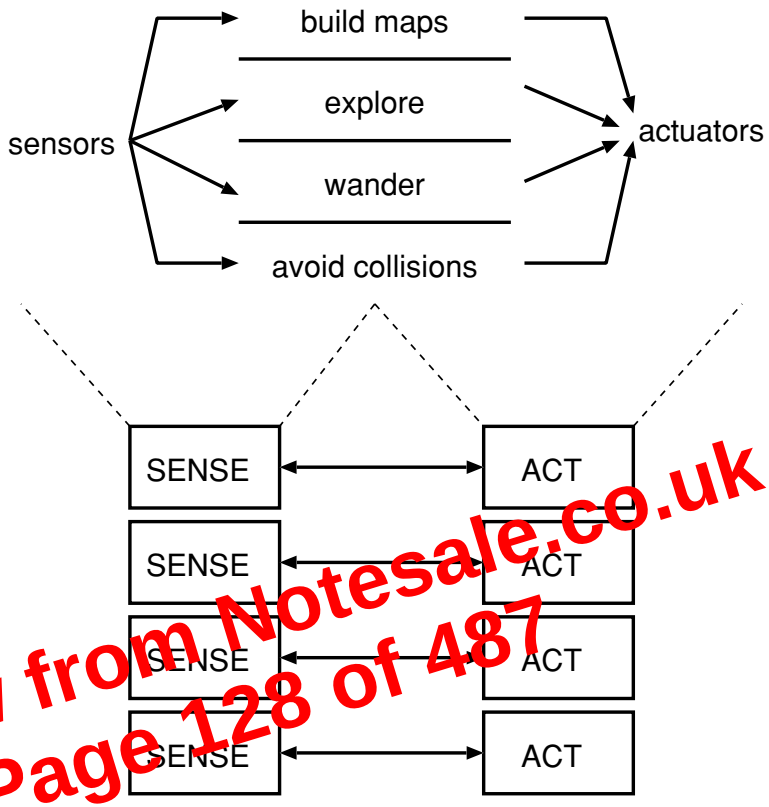


Figure 4.2 Vertical decomposition of tasks into an S-A organization, associated with the Reactive Paradigm.

Arkin and Payton used a potential fields methodology, favoring software implementations. Both approaches are equivalent. The Reactive Paradigm was initially met with stiff resistance from traditional customers of robotics, particularly the military and nuclear regulatory agencies. These users of robotic technologies were uncomfortable with the imprecise way in which discrete behaviors combine to form a rich emergent behavior. In particular, reactive behaviors are not amenable to mathematical proofs showing they are sufficient and correct for a task. In the end, the rapid execution times associated with the reflexive behaviors led to its acceptance among users, just as researchers shifted to the Hybrid paradigm in order to fully explore layering of intelligence.

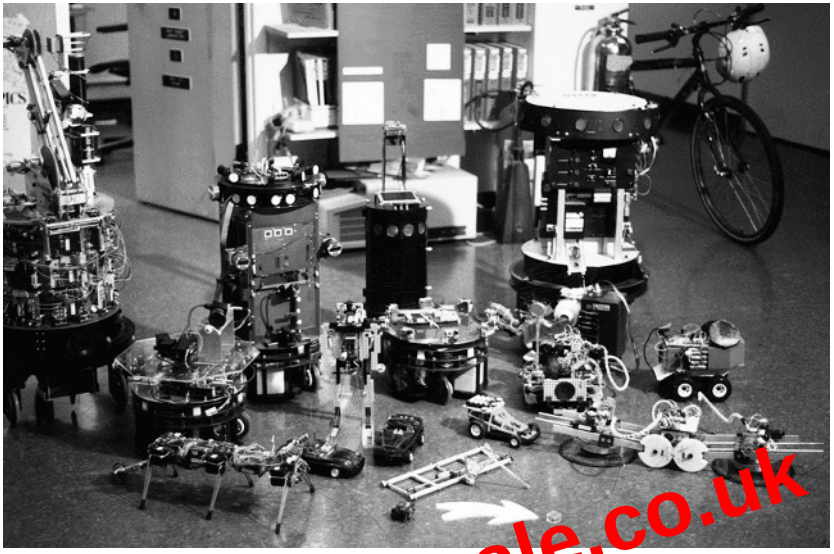


Figure 4.5 "Veteran" robots of the MIT AI Laboratory using the subsumption architecture. (Photograph courtesy of the MIT Artificial Intelligence Laboratory.)

to walk, avoid collisions, and climb over obstacles without the "move-think-move-think" pauses of Shakey.

The term "behavior" in the subsumption architecture has a less precise meaning than in other architectures. A behavior is a network of sensing and acting modules which accomplish a task. The modules are augmented finite state machines AFSM, or finite state machines which have registers, timers, and other enhancements to permit them to be interfaced with other modules. An AFSM is equivalent to the interface between the schemas and the coordinated control strategy in a behavioral schema. In terms of schema theory, a subsumption behavior is actually a collection of one or more schemas into an abstract behavior.

Behaviors are released in a stimulus-response way, without an external program explicitly coordinating and controlling them. Four interesting aspects of subsumption in terms of releasing and control are:

LAYERS OF COMPETENCE

1. Modules are grouped into *layers of competence*. The layers reflect a hierarchy of intelligence, or competence. Lower layers encapsulate basic survival functions such as avoiding collisions, while higher levels create

The use of layers and subsumption allows new layers to be built on top of less competent layers, without modifying the lower layers. This is good software engineering, facilitating modularity and simplifying testing. It also adds some robustness in that if something should disable the Level 1 behaviors, Level 0 might remain intact. The robot would at least be able to preserve its self-defense mechanism of fleeing from approaching obstacles.

Fig. 4.10 shows Level 1 recast as behaviors. Note that FEELFORCE was used by both RUNAWAY and AVOID. FEELFORCE is the perceptual component (or schema) of both behaviors, with the AVOID and RUNAWAY modules being the motor component (or schema). As is often the case, behaviors are usually named after the observable action. This means that the behavior (which consists of perception and action) and the action component have the same name. The figure does not show that the AVOID and RUNAWAY behaviors share the same FEELFORCE perceptual schema. As will be seen in the next chapter, the object-oriented properties of schema theory facilitate the reuse and sharing of perceptual and motor components.

LEVEL 2: FOLLOW CORRIDORS

Now consider adding a third layer to permit the robot to move down corridors, as shown in Fig. 4.11. (The third layer in Brooks' original paper is "explore," because he was considering a mapping task.) The LOOK module examines the sonar polar plot and identifies a corridor. (Note that this is another example of behaviors sharing the same sensor data but using it locally for different purposes.) Because identifying a corridor is more computationally expensive than just extracting range data, LOOK may take longer to run than behaviors at lower levels. LOOK passes the vector representing the direction to the middle of the corridor to the STAYINMIDDLE module. STAYINMIDDLE subsumes the WANDER module and delivers its output to the AVOID module which can then swerve around obstacles.

But how does the robot get back on course if the LOOK module has not computed a new direction? In this case, the INTEGRATE module has been observing the robots actual motions from shaft encoders in the actuators. This gives an estimate of how far off course the robot has traveled since the last update by LOOK. STAYINMIDDLE can use the dead reckoning data with the intended course to compute the new course vector. It serves to fill in the gaps in mismatches between updates rates of the different modules. Notice that LOOK and STAYINMIDDLE are quite sophisticated from a software perspective.

INTEGRATE is an example of a module which is supplying a dangerous internal state: it is actually substituting for feedback from the real world. If for some reason, the LOOK module never updates, then the robot could op-

describe here, so instead a generalization will be presented. Potential field styles of behaviors always use *vectors* to represent behaviors and *vector summation* to combine vectors from different behaviors to produce an emergent behavior.

VECTORS
VECTOR SUMMATION

4.4.1 Visualizing potential fields

The first tenet of a potential fields architecture is that the motor action of a behavior must be represented as a potential field. A potential field is an array, or field, of vectors. As described earlier, a vector is a mathematical construct which consists of a magnitude and a direction. Vectors are often used to represent a force of some sort. They are typically drawn as an arrow, where the length of the arrow is the magnitude of the force and the angle of the arrow is the direction. Vectors are usually represented with a boldface capital letter, for example, **V**. A vector can also be written as a tuple (m, d), where m stands for magnitude and d for direction. By convention the magnitude is a real number between 0.0 and 1, but the magnitude can be any real number.

ARRAY REPRESENTING A FIELD

The array represents a region of space. In most robotic applications, the space is in two dimensions, representing a bird's eye view of the world just like a map. The map can be divided up into squares, creating a (x,y) grid. Each element of the array represents a square of space. Perceivable objects in the world exert a force field on the surrounding space. The force field is analogous to a magnetic or gravitation field. The robot can be thought of as a particle that has entered the field exuded by an object or environment. The vector in each element represents the force, both the direction to turn and the magnitude or velocity to head in that direction, a robot would feel if it were at that particular spot. Potential fields are continuous because it doesn't matter how small the element is; at each point in space, there is an associated vector.

Fig. 4.12 shows how an obstacle would exert a field on the robot and make it run away. If the robot is close to the obstacle, say within 5 meters, it is inside the potential field and will fell a force that makes it want to face directly away from the obstacle (if it isn't already) and move away. If the robot is not within range of the obstacle, it just sits there because there is no force on it. Notice that the field represents what the robot should do (the motor schema) based on if the robot perceives an obstacle (the perceptual schema). The field isn't concerned with how the robot came to be so close to the obstacle; the robot feels the same force if it were happening to move within range or if it was just sitting there and someone put their hand next to the robot.

multiple range sensors? Bigger obstacles will be detected by multiple sensors at the same time. The common way is to have a RUNAWAY behavior for each sensor. This called multiple instantiations of the same behavior. Below is a code fragment showing multiple instantiations; all that had to be done is add a for loop to poll each sensor. This takes advantage of two properties of vector addition: it is associative (a+b+c+d can be performed as $((a+b)+c)+d$), and it is commutative (doesn't matter what order the vectors are summed).

```
while (robot==ON) {
   vector.mag=vector.dir=0.0; //initialize to 0
   for (i=0; i<=numberIR; i++) {
     vectorCurrent=Runaway(i); // accept a sensor number
     vectorOutput = VectorSum(tempVector,vectorCurrent);
   }
   turn(vector.direction);
   forward(vector.magnitude*MAX_VELOCITY);
}
```


BOX CANYON

As seen in Fig. 4.19, the robot is able to get out of the cave-like trap called a *box canyon* without building a model of the wall. Each instance contributes a vector, a pair of which have a X or Y component that cancels out.

From an ethological perspective, the above program is elegant because it is equivalent to behavioral instantiations in animals. Recall from Ch. 3 the model of *rana computrix* and its real-life toad counterpart where each eye sees and responds to a fly independently of the other eye. In this case, the program is treating the robot as if it had 8 independent eyes!

From a robotics standpoint, the example illustrates two important points. First, the direct coupling of sensing to action works. Second, behavioral programming is consistent with good software engineering practices. The RUNAWAY function exhibits *functional cohesion*, where the function does one thing well and every statement in the function has something directly to do with the function's purpose.[122] Functional cohesion is desirable, because it means the function is unlikely to introduce side effects into the main program or be dependent on another function. The overall organization shows *data coupling*, where each function call takes a simple argument.[122] Data coupling is good, because it means all the functions are independent; for example, the program can be easily changed to accommodate more IRs sensors.

FUNCTIONAL COHESION

DATA COUPLING

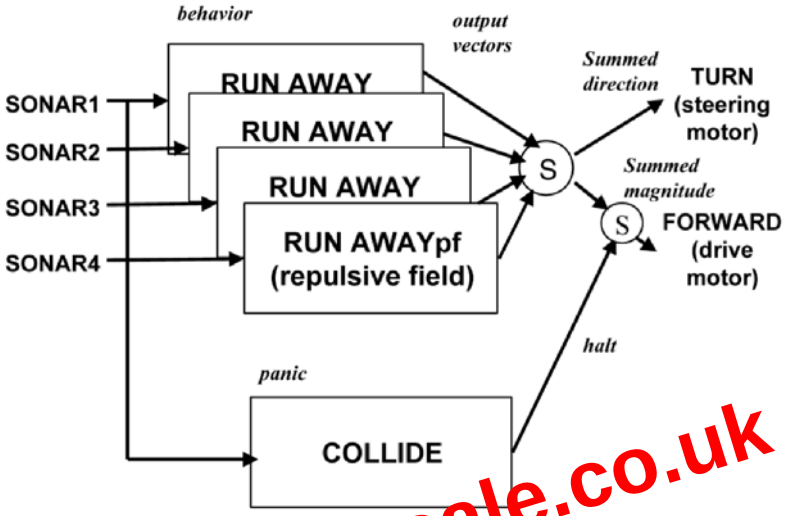


Figure 4.20 Level 1 redone as Potential Fields Methodology.

to move every n seconds. This would be represented by a uniform field where the robot felt the same attraction to go a certain direction, regardless of location, for n seconds. However, by combining the output of `WANDER` with the output vectors from `RUNAWAYpf`, the need for a new `AVOID` behavior is eliminated. The `WANDER` vector is summed with the repulsive vectors, and as a result, the robot moves both away from the obstacles and towards the desired direction. This is shown in Fig. 4.22. The primary differences in this example are that potential fields explicitly encapsulate sensing and acting into primitive behaviors, and it did not have to subsume any lower behaviors. As with subsumption, the robot became more intelligent when the `WANDERpf` behavior was added to the `RUNAWAYpf` behavior.

Now consider how Level 3, corridor following, would be implemented in a potential field system. This further illustrates the conceptual differences between the two approaches. The robot would have two concurrent behaviors: `RUNAWAYpf` and `follow-corridor`. `RUNAWAYpf` would remain the same as before, but `WANDER` would be discarded. In the parlance of potential fields, the task of following a corridor requires only two behaviors, while the task of wandering requires two different behaviors.

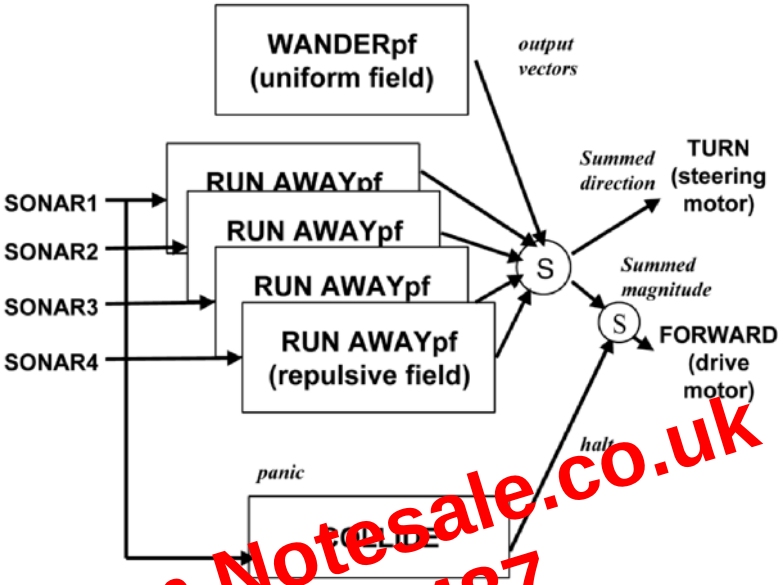


Figure 4.21 Level 1 reconcile with Potential Fields Methodology.

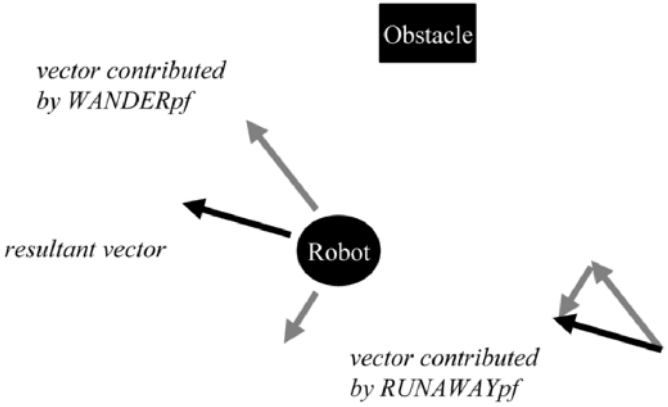


Figure 4.22 Example resultant vector of `WANDERpf` and `RUNAWAYpf`.

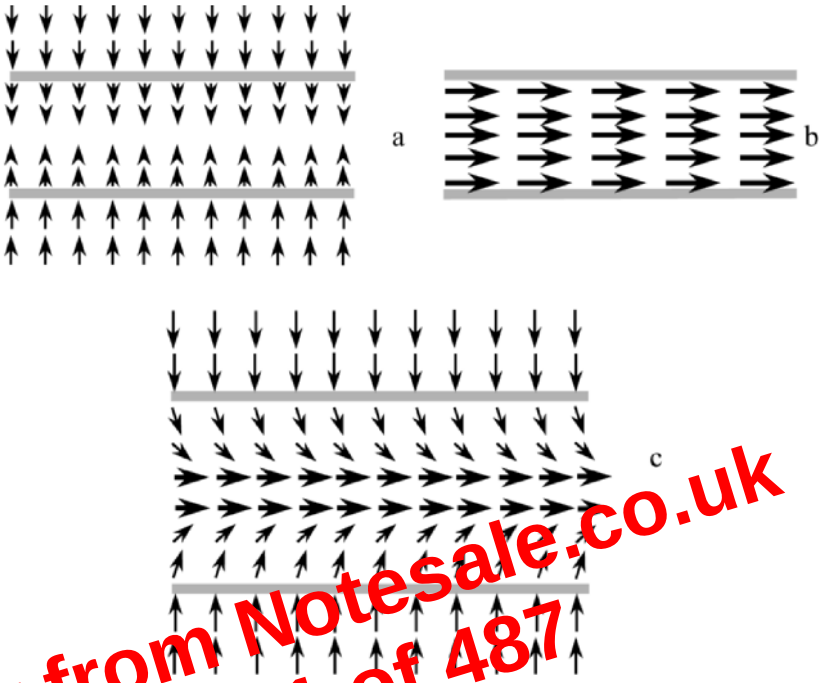


Figure 4.23 The a.) perpendicular and b.) uniform fields combining into c.) a `follow-corridor` field.

The `follow-corridor` behavior is interesting, because it requires a more complex potential field. As shown in Fig. 4.23, it would be desirable for the robot to stay in the middle of the corridor. This can be accomplished using two potential fields: a uniform field perpendicular to the left boundary and pointing to the middle, and a uniform field perpendicular to the right boundary and pointing to the middle. Notice that both fields have a linear decrease in magnitude as the field nears the center of the corridor. In practice, this taper prevents the robot from see-sawing in the middle.

Also notice that the two uniform fields are not sufficient because they do not permit the robot to move forward; the robot would move to the middle of the corridor and just sit there. Therefore, a third uniform field is added which is parallel to the corridor. All three fields combined yield a smooth field which sharply pushes the robot back to the middle of the corridor as a function of its proximity to a wall. In the meantime, the robot is constantly making forward progress. The figure below shows the fields involved. Re-

It is better software engineering to write a general move to goal behavior, where only what is the goal—a red region or a blue region—varies. The goal for the current instance can be passed in at instantiation through the object constructor.

Writing a single generic behavior for `move_to_goal(color)` is more desirable than writing a `move_to_red` and a `move_to_blue` behaviors. From a software engineering perspective, writing two behaviors which do the same thing is an opportunity to introduce a programming bug in one of them and not notice because they are supposed to be the same. Generic behaviors also share the same philosophy as factoring in mathematics. Consider simplifying the equation $45x^2 + 90x + 45$. The first step is to factor out any common term to simplify the equation. In this case, 45 can be factored and the equation rewritten as $45(x + 1)^2$. The color of the goal, red or blue, was like the common coefficient of 45; it is important, but tends to hide that the key to the solution was the move-to-goal part, or x.

Modular, generic code can be handled nicely by schemas as shown in Fig. 5.2. The behavior `move_to_goal` would consist of a perceptual schema, which will be called `extract-goal`, and a motor schema, which uses an attractive field called `pfields.attract()`. `extract-goal` uses the affordance of color to extract where the goal is in the image, and then computes the angle to the center of the colored region and the size of the region. This information forms the percept of the goal; the affordance of the Coke can is the color, while the information extracted from the perception is the angle and size. The attraction motor schema takes that percept and is responsible for using it to turn the robot to center on the region and move forward. It can do this easily by using an attractive field, where the larger the region, the stronger the attraction and the faster the robot moves.

The `move_to_goal` behavior can be implemented as a primitive behavior, where `goal_color` is a numerical means of representing different colors such as `red` and `blue`:

`move_to_goal(goal_color):`

Object	Behavioral Analog	Identifier
Data	percept	`goal_angle` `goal_strength`
Methods	`perceptual_schema` `motor_schema`	`extract_goal(goal_color)` `pfields.attraction(goal_angle, goal_strength)`

The above table implies some very important points about programming with behaviors:

the competition favored an entry which could complete the course without accruing any penalties over a faster entry which might drift over a boundary line or bump an obstacle. Entrants were given three runs on one day and two days to prepare and test on a track near the course; the times of the heats were determined by lottery.

Step 2: Describe the robot. The purpose of this step is to determine the basic physical abilities of the robot and any limitations. In theory, it might be expected that the designer would have control over the robot itself, what it could do, what sensors it carries, etc. In practice, most roboticists work with either a commercially available research platform which may have limitations on what hardware and sensors can be added, or with relatively inexpensive kit type of platform where weight and power restrictions may impact what it can reasonably do. Therefore, the designer is usually handed some fixed constraints on the robot platform which will impact the design.

In this case, the competition stated that the robot vehicle had to have a footprint of at least 3ft by 3.5ft but no bigger than a golf cart. Furthermore, the robot had to carry its own power supply and do all computing on board (no radio communication with an off-board processor was permitted) plus carry a 20 pound payload.

The CSM team was donated the materials for a robot platform by Omnitech Robotics, Inc. Fig. 5.4 shows Omnibot. The vehicle base was a Power Wheels battery powered children's jeep purchased from a toy store. The base met the minimum footprint exactly. It used Ackerman (car-like) steering, with a drive motor powering the wheels in the rear and a steering motor in the front. The vehicle had a 22° turning angle. The on-board computing was handled by a 33MHz 486 PC using Omnitech CANAMP motor controllers. The sensor suite consisted of three devices: shaft encoders on the drive and steer motors for dead reckoning, a video camcorder mounted on a mast near the center of the vehicle and a panning sonar mounted below the grille on the front. The output from the video camcorder was digitized by a black and white framegrabber. The sonar was a Polaroid lab grade ultrasonic transducer. The panning device could sweep 180°. All coding was done in C++.

Due to the motors and gearing, Omnibot could only go 1.5 mph. This limitation meant that it could only win if it went farther with less penalty points than any other entry. It also meant that the steering had to have at least a 150ms update rate or the robot could veer out of bounds without ever perceiving it was going off course. The black and white framegrabber eliminated the use of color. Worse yet, the update rate of the framegrabber was almost 150ms; any vision processing algorithm would have to be very fast or else the robot would be moving faster than it could react. The reflections from uneven grass reduced the standard range of the sonar from 25.5 ft to about 10 ft.

Documentaries.
Scientific American Frontiers did an excellent special on robot competitions called "Robots Alive!" The special covered the AUVS Aerial Vehicle Competition (take away lesson: try your robot outdoors before you show up at an outdoor robot competition) and the 1996 AAAI Mobile Robot Competition where the robots picked up orange tennis balls instead of coca-cola cans.

However, they wouldn't necessarily be equivalent in performance or update rate. As will be seen in this chapter, the sonar is liable to produce a noisy percept in a second or two, while stereo vision may take minutes. Even different stereo vision algorithms may produce different results on the same data stream. Therefore, the logical sensor contains a selector function which specifies the conditions under which each alternative is useful and should be selected.

Notice that a logical sensor can be implemented as a perceptual schema, where the methods are the alternative means of generating the percept and the coordinated control strategy contains the knowledge as to when a particular method is appropriate. Also note that each individual method can be implemented as a perceptual schema, leading to the recursive, building-block effect.

In reactive systems, the term logical sensor has degenerated somewhat from its original usage and is essentially equivalent to a perceptual schema. "Logical sensor" is often used to connote information hiding, where the particular sensor and processing algorithms is hidden in the "package." This is useful because a robot might use the same physical sensor in two different ways. An avoid behavior might use a polar plot of sonar range data, while a panic-stop behavior might use the minimum of all the incoming sonar data. Since the perceptual schema use the raw sonar data differently, it is as if they were different sensors.

6.2 Behavioral Sensor Fusion

SENSOR FUSION

Sensor fusion is a broad term used for any process that combines information from multiple sensors into a single percept. The motivation for sensor fusion stems from three basic combinations of sensors: *redundant* (or competing), *complementary*, and *coordinated*. Although many researchers treat sensor fusion as a means of constructing a global world model in a hierarchical or deliberative system, sensor fusion can be incorporated into behaviors through *sensor fission*, *action-oriented sensor fusion*, and *sensor fashion*.

REDUNDANT
COMPLEMENTARY
COORDINATED

In some cases multiple sensors are used when a particular sensor is too imprecise or noisy to give reliable data. Adding a second sensor can give another "vote" for the percept. When a sensor leads the robot to believe that a percept is present, but it is not, the error is called a *false positive*. The robot has made a positive identification of percept, but it was false. Likewise, an error where the robot misses a percept is known as a *false negative*. Sensors

FALSE POSITIVE

FALSE NEGATIVE

LOCOMOTION LOAD

platform. The power needed to move the robot is called the *locomotion load*. Unfortunately, many robot manufacturers focus on only the locomotion load, balancing power needs with the desire to reduce the overall weight and size. This leads to a very small hotel load, and often prevents many sensors from being added to platform.

5. Hardware reliability. Sensors often have physical limitations on how well they work. For example, Polaroid sonars will produce incorrect range reading when the voltage drops below 12V. Other sensors have temperature and moisture constraints which must be considered.

6. Size. The size and weight of a sensor does affect the overall design. A microrover on the order of a shoebox will not have the power to transport a large camera or camcorder, but it may be able to use a miniature "QuickCam" type of camera.

The above list concentrated on considerations for the physical aspects of the sensor. However, the sensors only provide observations; without the software perceptual schemas, the behaviors cannot use the sensors. Therefore, the software that will process the information from a sensor must be considered as part of the sensor selection process. **7. Computational complexity.** Computational complexity is the estimate of how many operations an algorithm or program performs. It is often written as a function O, called the "order," where $O(x)$ means the number of operations is proportional to x. x is often a function itself. Lower orders are better. An algorithm that executes with $O(n)$ computationally consuming operations is faster than one with $O(n^2)$ operations. (If you doubt this, see if you can find a positive, whole number value of n such that $n > n^2$.) Computational complexity has become less critical for larger robots, with the rapid advances in processors and miniaturization of components. However, it remains a serious problem for smaller vehicles.

8. Interpretation reliability. The designer should consider how reliable the sensor will be for the ecological conditions and for interpretation. The robot will often have no way of determining when a sensor is providing incorrect information. As a result the robot may "hallucinate" (think it is seeing things that are not there) and do the wrong thing. Many sensors produce output which are hard for human to interpret without years of training; medical X-rays are one example, and synthetic aperature radar (SAR) which produces polar plots is another. If a sensor algorithm was not working properly in these modalities, the designer might not be skilled enough to notice it. Therefore, the algorithms themselves must be reliable.

a.

b.

Figure 6.10 CRISbot, the Colorado School of Mines' entry in the 1997 AAAI Mobile Robot Competition Hors d'Oeuvres, Anyone? event. a.) Interacting with the audience. b.) Audience members "communicated" by kicking the two protruding bumpers near the bottom of the robot. (Photographs courtesy of AAAI.)

However, they are often interchangeable with IR sensors because IR sensors often operate over the short range (inches) with less reliability.

6.6 Computer Vision

COMPUTER VISION
IMAGE

Computer vision refers to processing data from any modality which uses the electromagnetic spectrum which produces an image. An *image* is essentially a way of representing data in a picture-like format where there is a direct physical correspondence to the scene being imaged. Unlike sonar, which returns a single range reading which could correspond to an object anywhere within a 30° cone, an image implies multiple readings placed in a two-dimensional array or grid. Every element in the array maps onto a small region of space. The elements in image arrays are called *pixels*, a contraction

PIXELS

light, so the camera device can either have many frame buffers, which create a pipeline of images (but is expensive), or have a low frame rate.

FRAMEGRABBER

A *framegrabber* is a card which fits inside a computer, accepts analog camera signals and outputs the digitized results. The card has a software driver which allows the robot software to communicate with the board. Framegrabbers can produce a grayscale or a color digital image. In the early part of the 1990's, color-capable framegrabbers were prohibitively expensive, costing around $3,000 USD. Now color framegrabbers can be purchased from $300 to $500 USD, and TV tuners which can capture a single frame are available for $50 USD.

6.6.2 Grayscale and color representation

The framegrabber usually expresses the grayscale value of a pixel as an 8 bit number (1 byte of computer memory). This leads to 256 discrete values of gray, with 0 representing black and 255 representing white. (Remember, 256 values means 0...255.)

Color is represented differently. First, there are many different methods of expressing color. Some PC printers use a subtractive method, where cyan plus yellow make green. Most commercial devices in the U.S. use a NTSC (television) standard. Color is expressed as the sum of three measurements: red, green, and blue. This is simply abbreviated as *RGB*.

RGB

COLOR PLANES

RGB is usually represented as three *color planes*, or axes of a 3D cube as shown in Fig. 6.11. The cubic represents all possible colors. A specific color is represented by a tuple of three values to be summed: (R, G, B). Black is (0,0,0) or 0+0+0, or no measurements on any of the three color planes. White is (255, 255, 255). The pure colors of red, green, and blue are represented by (255,0,0), (0,255,0), and (0,0,255) respectively. This is the same as in color graphics.

Notice that the cube dimensions in the figure are 256 by 256 by 256, where 256 is the range of integers that can be expressed with 8 bits. Since there are three color dimensions, a manufacturer may refer to this as 24-bit color (3 x 8), to distinguish their framegrabber from ones which map color onto a linear grayscale. The 8-bit color model is what is used to colorize old black and white movies. There are only 256 values of color, which is quite limited, and the gray values are often ambiguous. The pixel values of a person's red lips might be 185, while their dark blue dress is also 185. A person may have to indicate which regions in each frame of the film where 185=red and 185=dark blue. 8-bit color is not often used for robots, unless the robot will

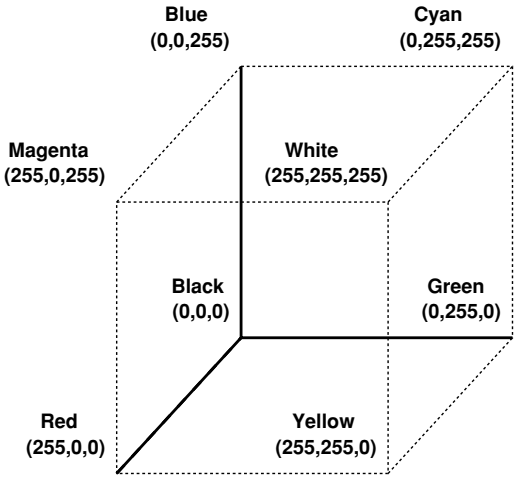


Figure 6.11 RGB color cube.

be operating in an environment where the only visible colors will not have an ambiguity.

24-bit color is usually sufficient for robotics. For other applications of computer vision such as medical imaging, weather forecasting, or military reconnaissance, an 8-bit resolution is often insufficient. Those applications may use 10 bits for each color plane. Since 10 bits do not fall on a byte boundary, it can be awkward to program algorithms for representing and manipulating this kind of image. Companies such as Pixar make special computers for these applications.

In programming terms, 24-bit color image is often declared in computer programs in two ways:

1. *Interleaved.* Interleaved means the colors are stored together, `RGB RGB RGB ...`, and it is the more common representation. The order is almost always red, green, then blue, although there may be framegrabbers which do not follow that convention. Below is a code fragment where the color is displayed for a pixel at location `row, col`.

```
#define    RED  0
#define    GREEN 1
#define    BLUE 2
```

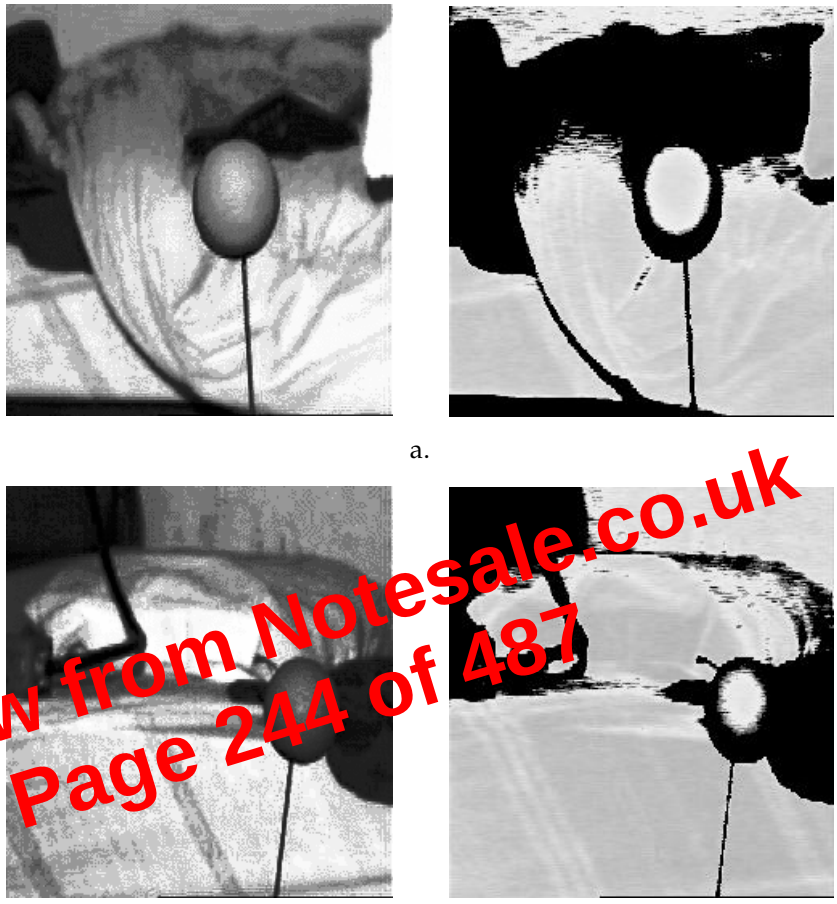


Figure 6.12 Images showing visual erosion of an orange landmark sticking up from a small robot (not visible): a.) Original image and RGB segmentation and b.) original image and degradation in RGB segmentation as robot moves farther from camera.

HSI
HUE
SATURATION
VALUE
INTENSITY

around the limitations of RGB. Such a device would work on the *HSI* (hue, saturation, intensity) representation of color. The *hue* is the dominant wavelength and does not change with the robot's relative position or the object's shape. *Saturation* is the lack of whiteness in the color; red is saturated, pink is less saturated. The *value* or *intensity* measure is the quantity of light received by the sensor. So HSV is a very different color scheme than RGB.

Figure 6.21 A stereo camera pair mounted on a pan/tilt head.

CAMERA CALIBRATION

cameras are perfectly matched optically and remain in alignment. In practice, robots move, bump, and suffer alignment drifts, plus the cameras may have some flaws in their optics. The alignment can be periodically compensated for in software through a *camera calibration* process, where the robot is presented with a standard and then creates a calibration look up table or function. Fig. 6.22 shows the CMU Uranus robot calibrating its camera system. As a result, many researchers are turning to units which package a stereo pair in a fixed case, where the alignment cannot be altered. Fig. 6.25 shows the results using a stereo range system using three cameras in a fixed configuration.

The first robot to use stereo vision successfully was Hans Moravec's Stanford Cart shown in Fig. 6.23a Moravec worked on the Cart while at graduate school at Stanford between 1973 and 1980. Fig. 6.23b shows the Marsokhod rover developed in the late 1990's which used a stereo pair for real-time navigation. Longer baselines tend to be more accurate because a slight measurement error has a smaller impact, but smaller baselines have a smaller "footprint," in effect, take up less room. The same point in both images still has to be identified.

Fig. 6.24 shows the simplified flow of operations in extracting range from a pair of images. The process begins with two images, the left-right pair, and results in a third image called the *range image* or the *depth map*. The left-right pair can be grayscale or color, but the depth map is a grayscale map, where intensity is proportional to the distance the pixel is away from the cameras. Fig. 6.25 shows two stereo images and the resulting depth map.

RANGE IMAGE
DEPTH MAP

Figure 6.28 Sick laser, covering a 180° area.

scanning component makes it are very expensive, on the order of $30,000 to $100,000 USD. A less expensive solution for navigation is to create a planar laser range finder.

A lidar produces two images: intensity and range. Fig. 6.27 shows the images produced by an Odetics laser range (LADAR) camera. The intensity map is essentially a black and white photograph and measures the intensity of the light reflected or absorbed by objects in the scene. This corresponds to how humans perceive the scene The image function for the range image is depth from the camera. Pixels that are black, or have a value of 0, are closer than white pixels. A flat floor usually appears as a radiating set of semi-circles going from near to far; trigonometry is then used to compute that the circles represent a flat surface. This process is called *range segmentation* and can be quite difficult.

RANGE SEGMENTATION

Lidars have some problems in practice. For example, Fig. 6.27 shows an area on the range image that is pure black or very near. But as can be seen from the intensity image the area is actually far away. Likewise, the black moulding between the wall and floor appear to be very far away on the range image. The errors were due to out of range conditions, absorption of the light (not enough light returned), or to the optical equivalent of specular reflection (light hitting corners gets reflected away from the receiver).

A planar laser range finder, such as the Sick shown in Fig. 6.28, provides a narrow horizontal range map. The map is essentially a high resolution polar

a.

b.

c.

d.

Figure 6.30 USF robots in the "Hors d'Oeuvres, Anyone?" event. a.) Family portrait, where Borg Shark is on the left, Puffer Fish on the right, with b.) the thermal sensor located as the Borg Shark's "third eye," c.) the SICK laser located behind the Borg Shark's teeth (head piece is removed for better view), and d.) a profile of Puffer Fish's skirt showing spatial relationship to sonar.

Step 1: Describe the task. The "Hors d'Oeuvres, Anyone?" event required fully autonomous robots to circulate in the reception area at the AAAI conference with a tray of finger food, find and approach people, interact with them, and refill the serving tray. Each robot was scored on covering the area, noticing when the tray needed refilling, interacting with people naturally, having a distinct personality, and recognizing VIPs. The USF entry used two robots, shown in Fig. 6.30, costumed by the USF Art Department in order to attract attention. The Borg Shark was the server robot, and navigated through audience following a pre-planned route. It would stop and serve at regular intervals or whenever a treat was removed from the tray. It used a DEC-talk synthesizer to broadcast audio files inviting audience members to remove a treat from its mouth, but it had no way of hearing and understanding natural language human commands. In order to interact more naturally with people, the Borg Shark attempted to maintain eye contact with people. If it saw a person, it estimated the location in image coordinates of where a VIP's colored badge might be, given the location of the face.

When the Borg Shark was almost out of food, she would call over radio ethernet her assistant robot, Puffer Fish. Puffer Fish would be stationary in sleep mode, inhaling and exhaling through her inflatable skirt and turning her camera as if avoiding people crowding her. When Puffer Fish awoke, she would head with a full tray of food (placed on her stand by a human) to the coordinates given to her by Borg Shark. She would also look for Borg Shark's distinctive blue costume, using both dead reckoning and visual search to move to goal. Once within 2 meters of Borg Shark, Puffer Fish would stop. A human would physically swap trays, then kick the bumpers to signal that the transfer was over. Borg Shark would resume its serving cycle, while Puffer Fish would return to its home refill station.

Both robots were expected to avoid all obstacles: tables, chairs, people. Since there was a tendency for people to surround the robotos, preventing coverage of the area or refilling, the robots had different responses. Borg Shark, who was programmed to be smarmy, would announce that it was coming through and begin moving. Puffer Fish, with a grumpy, sullen personality, would vocally complain, then loudly inflate her skirt and make a rapid jerk forward, usually causing spectators to back up and give her room.

Step 2: Describe the robots. The robots used for the entry were Nomad 200 bases, each with a unique sensor suite.

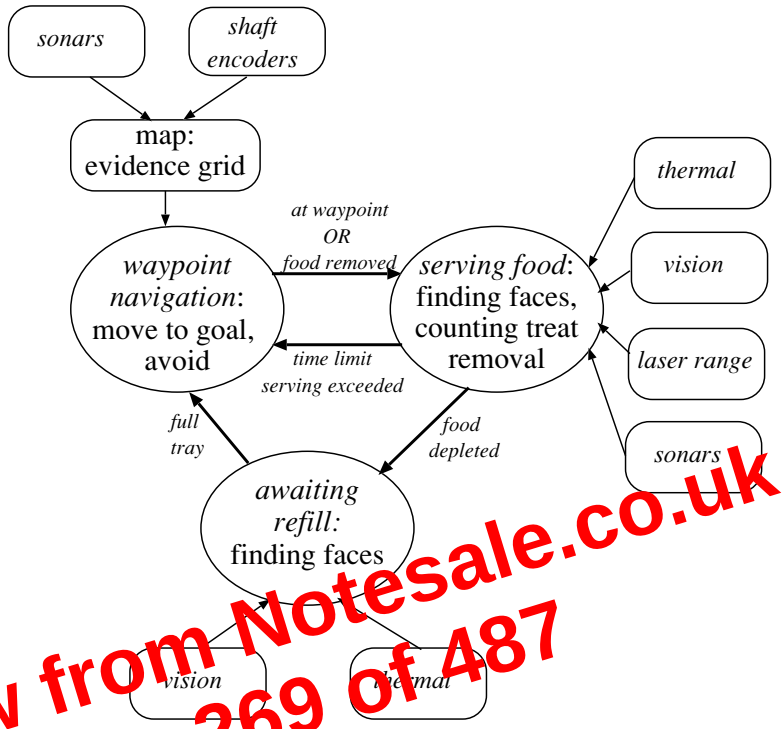


Figure 6.31 State diagrams for the Borg Shark, annotated to show sensors being used.

mining the temperature of a person on contact, but it was able to detect an increase in temperature above the ambient from a person up to two meters away.

Why wasn't the vision system simply replaced with the thermal sensor? This gets back to the attributes of sensors and how they fit with the environment. The thermal sensor has a 2° field of view, making it too slow to scan. Instead, the vision system covered a much wider field of view and could generate a small list of candidate regions. Then as the camera turned to center on the largest region, the thermal probe could decide whether this was really a person or not.

The problem with the food-count was greatly reduced by a simple `AND` function with the sonars. The system counted a treat as being removed only if there was a close range reading in front of the tray at the same time.

most common color coordinate systems are RGB and HSV. HSV treats color in absolute terms, but RGB is favored by equipment manufacturers. A color space used in biomedical imaging, SCT, appears to be less sensitive to lighting conditions than RGB and RGB-derived HSV. Many reactive robots exploit color as an affordance. This can be done by thresholding an image and identifying regions of the appropriate color. A color affordance method which works well for objects with multiple colors is color histogramming. Stereo range finding is an important class of algorithms for navigation, though the computational complexity has prevented it being ported to many mobile robot applications. Laser range finders, particularly the inexpensive planar rangers, have grown in popularity over the past few years.

Despite the diversity of sensors and affordances inherent in the environment, reactive robotics is remarkable for its lack of sophistication in sensing. This may stem from the split between computer vision and robotics in the formative years of the field. Many roboticists still assume algorithms developed by computer vision specialists are too computationally expensive to work on commercially available onboard processors. This is no longer true, in part because of the increased computational power of general purpose chips. Readers are encouraged to explore the large body of literature on computer vision and free tools on the web.

6.10 Exercises

Exercise 6.1

Define sensor suite, active/passive sensors, dead reckoning, computer vision. †

Exercise 6.2

Compare and contrast the RGB and HSV color representations specifying the advantages and disadvantages of each type. †

Exercise 6.3

Ultrasonic sensors have many positive and negative attributes. Name and describe three positive and three negative attributes. †

Exercise 6.4

What is the difference between physical and logical redundancy? †

Exercise 6.5

Describe the three major problems of ultrasonic sensing, and define a hypothetical instance in which a robot would encounter each problem (such as a room with a large amount of glass surfaces). †

7 *The Hybrid Deliberative/Reactive Paradigm*

Chapter objectives:

- Be able to describe the Hybrid Deliberative/Reactive paradigm in terms of i) sensing, acting, and planning and ii) sensing organization.
- Name and evaluate one representative hybrid architecture in terms of: support for modularity, niche targetability, ease of portability to other domains, robustness.
- Given a list of responsibilities, be able to say whether it belongs in the deliberative layer, or in the reactive layer.
- List the five basic components of a Hybrid architecture: sequencer agent, resource manager, cartographer, mission planner, performance monitoring and problem solving agent.
- Be able to describe the difference between managerial, state hierarchy, and model-oriented styles of Hybrid architectures.
- Be able to describe the use of state to define behaviors and deliberative responsibilities in state hierarchy styles of Hybrid architectures.

7.1 Overview

By the end of the 1980's, the trend in artificially intelligent robots was to design and program using the Reactive Paradigm. The Reactive Paradigm allowed robots to operate in real-time using inexpensive, commercially available processors (e.g., HC6811) with no memory. But the cost of reactivity, of course, was a system that eliminated planning or any functions which involved remembering or reasoning about the global state of the robot relative

mines what functionality goes in what modules, what modules have access to global knowledge (which leads to specifying public and friend classes in C++), and what that global knowledge (shared data structures) should be. Likewise, it is important to subdivide the deliberative portion into modules or objects. A good decomposition will ensure portability and reusability. While Hybrid architectures are most noteworthy for how they incorporate deliberation into mobile robotics, they also introduce some changes in the way reaction is organized. Many researchers found the two primary means of combining reactive behaviors—subsumption and potential field summation—to be limited. Since then at least three other mechanisms have been introduced: *voting* (in the DAMN architecture),[121] *fuzzy logic* (Saphira),[77] and *filtering* (SFX).[107]

The number of Hybrid architectures is rapidly increasing. This section attempts to introduce some conceptual organization on Hybrids in two ways. First, it offers a set of common components—essentially things to look for in a Hybrid architecture. Second, it divides Hybrids into three broad categories: *managerial, state hierarchies,* and *model oriented*.

7.3.1 Common components of hybrid architectures

While Hybrid architectures vary significantly in how they implement deliberative functionality, what they implement is fairly similar. Generally a Hybrid architecture has the following modules or objects:

SEQUENCER

- A *Sequencer* agent which generates the set of behaviors to use in order to accomplish a subtask, and determines any sequences and activation conditions. The sequence is usually represented as a dependency network or a finite state machine, but the sequencer should either generate this structure or be able to dynamically adapt it. Recall from Ch. 5 that assemblages of reactive behavior are manually constructed.

RESOURCE MANAGER

- A *Resource manager* which allocates resources to behaviors, including selecting from libraries of schemas. For example, a robot may have stereo vision, sonars, and IR sensors, all of which are capable of range detection. The behavioral manager would ascertain whether the IR sensors can detect at a sufficient range, the stereo vision can update fast enough to match the robot's desired velocity, and the sonars have enough power to produce reliable readings. In reactive architectures, the resources for a behavior were often hard-coded or hardwired, despite the ability of hu-

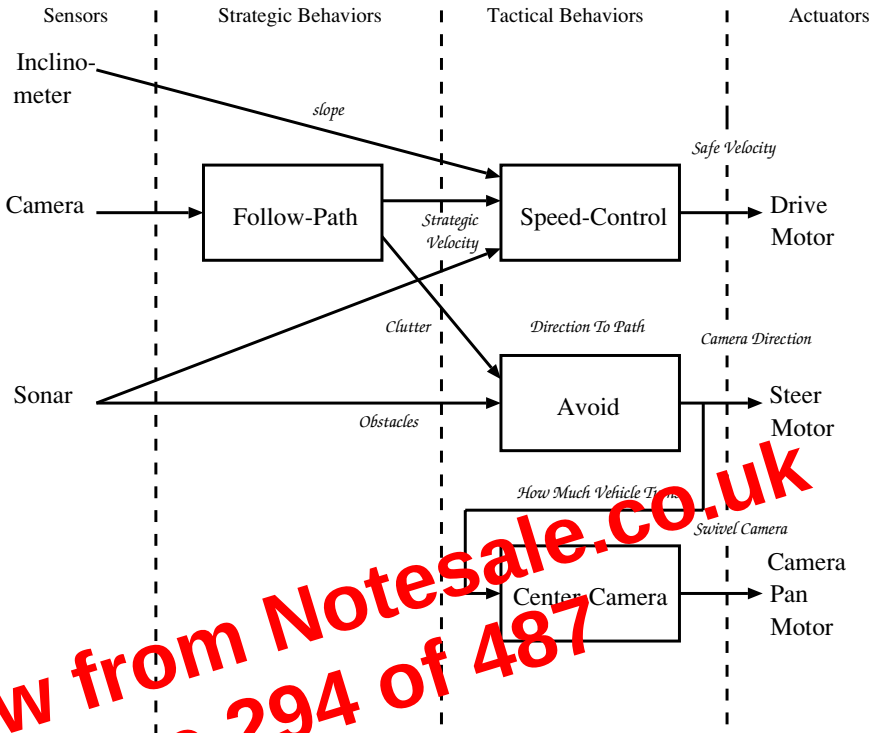


Figure 7.6 Strategic and tactical behaviors for following an outdoor path, used in the 1995 UGV Unmanned Ground Robot competition.

ensure that the robot acts in a safe manner in as close accordance with the strategic intent as possible. The interaction of strategic and tactical behaviors is still considered emergent behavior.

One outcome of the strategic-tactical partitioning was the discovery that every task to date could be done with one strategic behavior and several tactical behaviors. This means that the need to combine behaviors does not occur often, and so the combination mechanism is not particularly important. However, it should be emphasized that the strategic behaviors were often assemblages in the form of scripts. There were many strategic behaviors, but they were explicitly coordinated and controlled according to behavior-specific knowledge.

come by the large number of robots. The other approach, characterized by Carnegie Mellon University's Ambler robot, proposed a single large robot with a higher viewpoint and stability. Ambler was built by "Red" Whitaker at the CMU Field Robotics Institute to be able to maintain a sensor platform at a level height by stepping over the majority of rocks, but at a tremendous penalty in size, weight, and power. In the end, planetary rover researchers have gravitated towards wheeled vehicles with some type of articulation to maintain stability, such as seen with Sandia National Laboratories' Rattler. An extension of the Ambler design philosophy was manifested in the Dante robots. These were built to rappel down steep canyons and volcanoes on Mars (and Earth). Dante was able to lower itself successfully most of the way into a volcano in Antarctica, but could not climb back out. It was dropped while being lifted out by a helicopter, twisting its frame.

7.8 Evaluation of Hybrid Architectures

In some regards, it is difficult to evaluate Hybrid architectures individually. Each architecture is still evolving and the deliberative component is being expanded practically daily. Returning to the four criteria set forth in the overview in Part I, it is interesting to evaluate the architectures as a whole.

In terms of support for modularity, each architecture is highly modular. Most are divided into layers, which are then subdivided into modules. As the software agent programming style for AI gains in popularity, probably more architectures will implement deliberative modules as independent specialists. AuRA and SFX clearly exhibit an organization which lends itself to object-oriented programming. The use of specialists also enhances the ease of portability.

Hybrids tend to have a high degree of niche targetability. The addition of the deliberative component allows Hybrids to be used for applications not appropriate for purely Reactive systems. However, the partitioning between reaction and deliberation allows the reactive portion to be used alone for purely reactive applications.

Another attractive aspect of Hybrid architectures is that they often explicitly attempt to ensure robustness. In the SFX and 3T architecture, modules within the various deliberative components attempt to monitor the performance of the reactive behaviors and either replace or adapt the configuration as needed.

no awareness of the goals of the other team members.

Now consider what happens when a robot ant encounters an asteroid it can't move. The robot stays there pushing. Eventually another robot will come along because the asteroid is not moving. As it is attracted to the "dark side" of the asteroid, it will come into range of the first robot. What happens? The avoid-robot behavior should be instantiated, causing the first robot to move over a bit. The second robot will also feel a repulsive force and slow down. As the first robot moves out of the way, the angle of repulsion changes, forcing the second robot to move sideways as well, as it continues to move to the asteroid. Together, the interaction between the two robots should cause them to naturally balance themselves behind the asteroid and push together. The point is that the robots were not explicitly directed to all work on the same NEO; they were each directed to find their own NEO, but circumstances led them to the same one.

8.6 Emergent Social Behavior

The examples of heterogeneity, cooperation, control, and goals give some hint of how an overall social behavior emerges from the actions of autonomous robots. The robot teams often are the result of extensive design efforts, where the teams aren't too large to interfere with each other, and are optimally sized for the particular task, etc. Many researchers are exploring the issues of what happens when the designer doesn't have a choice about the size of the robot population. How do social behaviors emerge in those cases? And how can social rules or conventions be established to make the team self-regulating and productive? This section summarizes two approaches: creating social rules for the robots to follow, and allowing internal motivation to cause the robots to adapt their behavior to problems.

8.6.1 Societal rules

Maja Mataric has focused her research on how group dynamics might emerge in herds of multiple agents operating under fully distributed control. She explored the impact of density and the impact of societal rules on overall team performance.[90] Each IS Robotics R2 robot was programmed with behaviors using the Subsumption architecture. She set up a scenario where up to 20 identical robots (now known as "The Nerd Herd") were given the same location as a goal. The goal, however, was on the other side of a partition with a narrow door, permitting only one robot to pass through the partition at a

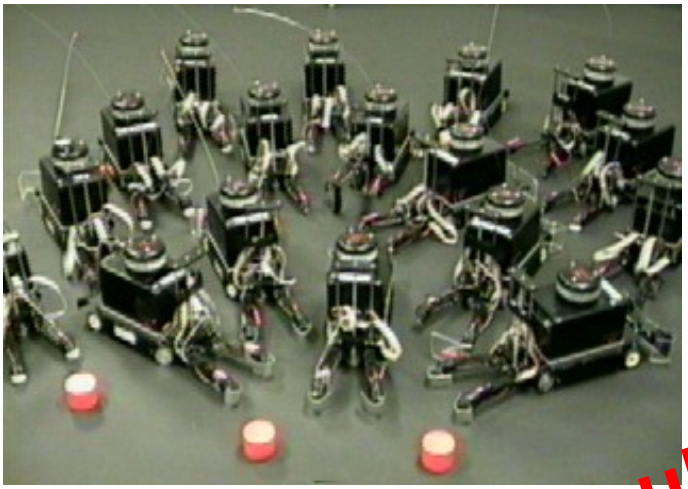


Figure 8.4 The Nerd Herd. (Photograph courtesy of USC Interaction Laboratory.)

time. The robots were placed randomly on the same side of the partition and started moving at the same time.

IGNORANT COEXISTENCE

In the first set of demonstrations, the robots functioned with *ignorant coexistence*. The robots coexisted in a team, but did not have any knowledge of each other. A robot treated another robot as an obstacle. Each robot had the equivalent of a move-to-goal and an avoid-obstacle behavior. Since robots were treated as obstacles, once the robots gathered at the opening, they spent most of their time avoiding each other. The team as a whole made slow progress through the door to the goal location. Worse yet, the larger the number of robots fielded, the larger the traffic jam, and the longer to get all the team members through.

INFORMED COEXISTENCE

In the second demonstration, *informed coexistence*, the robots were allowed to recognize each other and given a simple social rule governing inter-robot interactions. In addition to move-to-goal and avoid-obstacle, a third behavior was created for avoiding robots. If a robot detected another robot, it would stop and wait for time p. If the blocking robot was still in the way after p, the robot would turn left and then resume moving to the goal. The result of the new behavior was to reduce the traffic jams, and the group got through the door in about the same time as a single agent going back and forth through the opening 20 times.

INTELLIGENT COEXISTENCE

The real surprise came in the third demonstration, *intelligent coexistence*.

time	Robot 1	Robot 2
0	find-stationary-asteroid	find-stationary-asteroid
1	sees A1	sees A1
2	move-to-asteroid(A1)	move-to-asteroid(A1)
3	arrives at A1	resumes find-stationary-asteroid
4	push-asteroid-to-home(A1) T1-acquiescence++	find-stationary-asteroid T1-impatience++
5	push-asteroid-to-home(A1) T1-acquiescence++	sees A2 T1-impatience++
6	push-asteroid-to-home(A1) T1-acquiescence++	move-to-asteroid(A2) T1-impatience++
7	push-asteroid-to-home(A1) T1-acquiescence++ T2-impatience++	push-asteroid-to-home(A2) T1-impatience>limit put T1 on stack T2-acquiescence++
8	push-asteroid-to-home(A1) T1-acquiescence++ T2-impatience++	push-asteroid-to-home(A2) A1-impatience++ T2-acquiescence++
9	T1-acquiescence>limit gives up on T1 find-stationary-asteroid T2-impatience++	push-asteroid-to-home(A2) T2-acquiescence++
10	T2-impatience>limit no attempts T2 move-to-asteroid(A2)	T2-acquiescence++
11	push-asteroid-to-home(A2) T2-acquiescence = 0	push-asteroid-to-home(A2) T2-acquiescence = 0
12	arrives at HOME	arrives at HOME

Figure 8.6 Example of how the internal motivation in ALLIANCE might be extended to work with two space ants.

8.7 Summary

In summary, many tasks favor the use of many cheap robots rather than a single expensive one. These collections of multiple robots are often referred to as multi-agents. Individual robots in a multi-agent team are generally programmed with behaviors, most often as purely reactive systems, but occasionally with a hybrid architecture. As with an overall behavior emerging

digm. Most of the techniques presented in Part II will go into the deliberative component of Hybrid architectures.

One important observation is that the four questions of navigation largely ignore an implicit fifth question: *how am I going to get there?* Based on Part I, the obvious answer is "by using reactive behaviors." But navigation is deliberative, and the issue of integrating deliberation and reaction for navigation in a Hybrid architecture is still largely open. Work addressing this issue of interleaving planning and execution is presented in Ch. 9.

Spatial Memory

The answer to *what's the best way there?* depends on the representation of the world that the robot is using. The world representation will be called the robot's *spatial memory*.[63] Spatial memory is the heart of the cartographer object class (or its equivalent) in a Hybrid architecture, as described in Ch. 7.

SPATIAL MEMORY

Spatial memory should provide methods and data structures for processing and storing output from current sensory inputs. For example, suppose a robot is directed to "go down the hall to the third red door on the right." Even for the coordination and control of reactive behaviors, the robot needs to operationalize concepts such as "hall," "red," "door" into features to look for with a perceptual schema. It also needs to remember how many red doors it has gone past (and not count the same door twice!). It would also be advantageous if the robot sensed a barrier or dead-end and updated its map of the world.

Spatial memory should also be organized to support methods which can extract the relevant expectations about a navigational task. Suppose a robot is directed this time to to "go down the hall to the third door on the right." It could consult its spatial memory and notice that odd numbered doors are red, and even numbered are yellow. By looking for "red" and "yellow" in addition to other perceptual features of a door, the robot can more reliably identify doors, either by focus of attention (the robot only runs door detection on red and yellow areas, not every image) or by sensor fusion (more sources of data means a more certain percept).

Spatial memory supports four basic functions:

ATTENTION 1. *Attention.* What features, landmarks to look for next?

REASONING 2. *Reasoning.* Can that surface support my weight?

PATH PLANNING 3. *Path planning.* What is the best way through this space?

Figure 9.1 Artificial landmarks used in the 1992 AAAI Mobile Robot Competition. (Photograph courtesy of AAAI.)

9.3 Relational Methods

Relational methods represent the world as a graph or network of nodes and edges. Nodes represent gateways, landmarks, or goals. Edges represent a navigable path between two nodes, in effect that two nodes have a spatial relationship. Additional information may be attached to edges, such as direction (N,S,E,W), approximate distance, terrain type, or the behaviors needed to navigate that path. Paths can be computed between two points using standard graph algorithms, such as Dijkstra's single source shortest path algorithm. (See any algorithm textbook for details.)

One of the earliest investigations of relational graphs for navigation was by Smith and Cheeseman.[133] They represented the world as a relational graph, where the edges represented the direction and distance between nodes. They

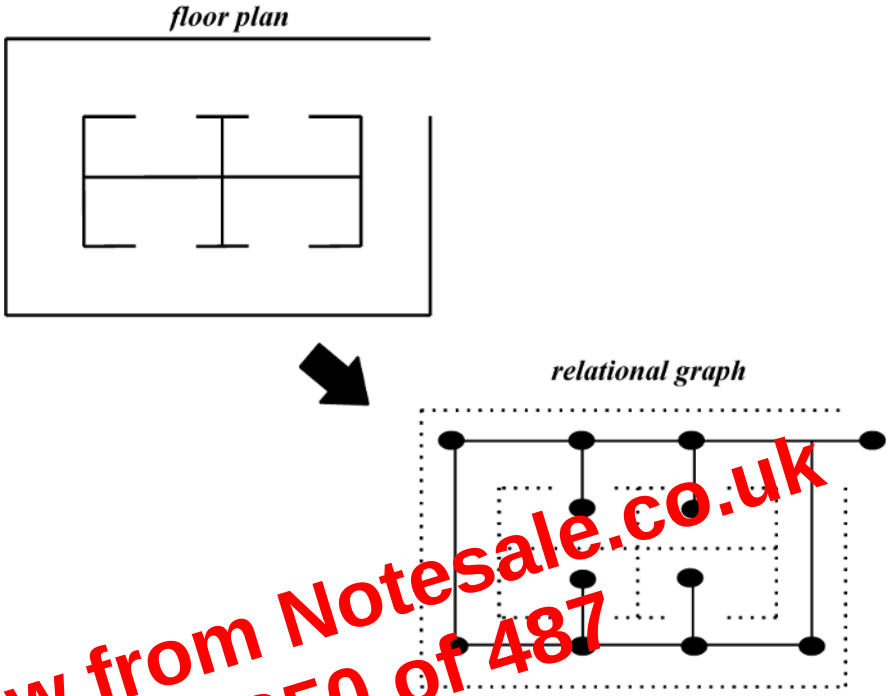


Figure 9.2 Representation of a floor plan as a relational graph.

simulated what would happen if a robot used dead-reckoning to navigate. As would be expected from the section on proprioception in Ch. 6, they found that the error would continually increase and soon the robot would be unable to reach any of the nodes.

9.3.1 Distinctive places

DISTINCTIVE PLACE

Kuipers and Byun tied relational graphs to sensing in their seminal work with distinctive places.[81] A *distinctive place* is a landmark that the robot could detect from a nearby region called a neighborhood. Their work was motivated by research in cognitive science indicating that spatial representation in the animal kingdom forms a multi-level hierarchy. (More recent studies suggest this hierarchy isn't as clearly partitioned as previously thought.) The lowest level, or most primitive way of representing space, was by identifying landmarks (doors, hallways) and the procedural knowledge to travel

rors could use multiple trips between nodes to build up a reasonable metric map, since most of the errors would average out. Another attractive aspect of the distinctive place approach is that it supports discovery of new landmarks as the robot explores an unknown environment. As long as the robot found something distinctive that it could reliably localize itself to, it could be put on a topological map. Then as it repeatedly moved to it, the robot could construct a metric map.

Returning to the discussion of landmarks, it should be noticed that a landmark must be unique to a node pair. There can't be any corners in the real world that are not on the graph between the nodes or else the robot will localize itself incorrectly.

The distinctive place approach as originally formulated encountered some problems when behavior-based roboticists began to apply it to real robots. One of the most challenging problems was perception. Good distinctive places are hard to come by; configurations that seemed useful to humans, like corners, proved difficult to reliably sense and localize against. Features that were easy to sense often were too numerous in the world, and so were not locally unique. Another challenge was learning the local control strategy. As the robot explored an unknown environment, it was easy to imagine that it could find distinctive places. But how did it learn the appropriate local control strategy? In a kind of environment, the robot might resort to always using wall following, even though other behaviors would be better suited. How would it ever try something different? Another open issue is the problem of indistinguishable locations. The issue of indistinguishable locations has also been tackled to some degree by work with probablistic methods, which will be covered in Ch. 11.

9.4 Associative Methods

ASSOCIATIVE METHODS

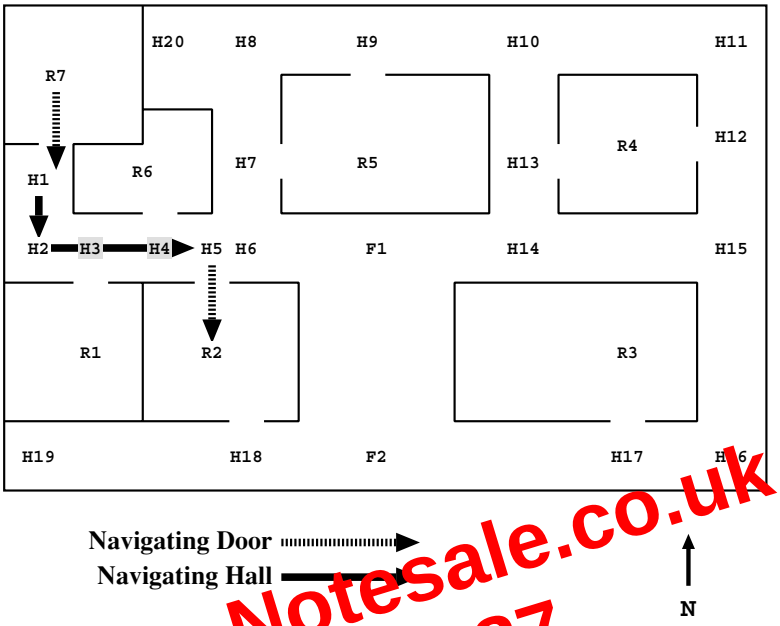
Associative methods for topological navigation essentially create a behavior which converts sensor observations into the direction to go to reach a particular landmark. The underlying assumption is that a location or landmark of interest for navigation usually has two attributes:

PERCEPTUAL STABILITY

1. *perceptual stability*: that views of the location that are close together should look similar

PERCEPTUAL DISTINGUISHABILITY

2. *perceptual distinguishability*: that views far away should look different.




```
R7 -> R2
R7 - H1 - H2 - H5 - R
Moving from R7 to H1, going SOUTH
In navigating door behavior
        ultra looking for door towards the: SOUTH
        MOVE AHEAD MOTOR ACTIVE
        Found door - Initialization terminated
        MOVE THROUGH DOOR MOTOR ACTIVE
Moved through door - Nominal Behavior terminated

Moving from H1 to H2, going SOUTH
In navigating hall behavior
        turning towards the: SOUTH
        Turned towards hall - Initialization terminated
        looking for hall towards the: EAST
        HALL FOLLOW MOTOR ACTIVE
Found hall - Nominal Behavior terminated

Moving from H2 to H5, going EAST
In navigating hall behavior
        turning towards the: EAST
        Turned towards hall - Initialization terminated
        vision looking for door relative: 90 (right side)
        HALL FOLLOW MOTOR ACTIVE
Found door (vision) - Nominal Behavior terminated

Moving from H5 to R2, going SOUTH
In navigating door behavior
        ultra looking for door towards the: SOUTH
        following wall on left (right ground truth)
        WALL FOLLOW MOTOR ACTIVE
        Found door - Initialization terminated
        MOVE THROUGH DOOR MOTOR ACTIVE
Moved through door - Nominal Behavior terminated
Goal location reached!
```


Figure 9.12 Scenario for moving from `R7` to `R2`. Shaded gateways are extraneous and discarded by the planner.

```
                if hall-not-found
                     wallfollow until find the hall
                else
                     if not facing hall
                          turn to face hall
           else starting in a HALL
                if not facing hall
                     turn to face hall
      case facing-hall:
           //nominal activity phase
           wallfollow until next gateway
```

The navigate-hall ANB terminates when the next expected gateway (the next node in the path) is found. There are three behaviors which look for gateways. hallwatch looks for the ultrasonic signature of a hall in the expected direction; foyerwatch similarly looks for foyers, and the robot uses vision through the confirm-door behavior to detect the landmark associated with the room. These behaviors run concurrently with the nominal behaviors.

The navigate-foyer ANB is used to move the robot between two foyers. It assumes that two foyers' nodes connected are a convenient representation of a single large foyer with entries from different directions (i.e., multiple gateways into the foyer). The script moves the robot n feet into the first foyer in the direction of the second foyer. This gets the robot away from potentially confusing ultrasonic signatures. Then the task manager determines which side of the foyer to wall-follow until the next expected gateway is detected. There is no case statement in the pseudocode since the sequence of activities is fixed.

```
//step 1
move-to-goal(n, dir) in direction of next foyer
//step 2
wallfollow until next gateway is detected
```

9.5.3 Lessons learned

The CSM team did not place at the AAAI competition; the robot suffered a series of hardware failures, causing both the power supply and the sonars to

landmarks in the world. Topological navigation focused on subgoals which are gateways or locations where the robot could change its primary heading.

The terms "optimal" and "best" have serious ramifications for robotics. In order to say a path is optimal, there is an implied comparison. As will be seen, some metric methods are able to produce an optimal path because they consider all possible paths between points. This can be computationally expensive. Fortunately, some algorithms (especially one named "A*" for reasons that will be discussed later) are more clever about rejecting non-optimal paths sooner than others.

Surprisingly, an optimal path may not appear optimal to the human eye; for example, a mathematically optimal path of a world divided into tiles or grids may be very jagged and irregular rather than straight. The ability to produce and compare all possible paths also assumes that the planning has access to a pre-exisiting (or *a priori*) map of the world. Equally as important, it assumes that the map is accurate and up to date. As such, metric methods are compatible with deliberation, while qualitative methods work well with more reactive systems. As a deliberative function, metric methods tend to be plagued by the same sorts of difficulties that were seen in Hierarchical systems: challenges in world representation, handling dynamic changes and surprises, and computation complexity.

COMPONENTS OF METRIC PATH PLANNERS

Metric path planners have two components: the *representation* (data structure) and the *algorithm*. Path planners first partition the world into a structure amenable to path planning. They use a variety of techniques to represent the world; no one technique is dominant, although regular grids appear to be popular. The intent of any representation is to represent only the salient features, or the relevant configuration of navigationally relevant objects in the space of interest; hence the term configuration space. Path planning algorithms generally work on almost any configuration space representation, although as with any algorithm, some methods work better on certain data structures. The algorithms fall into two broad categories: those which treat path planning as a graph search problem, and those which treat path planning as a graphics coloring problem. Regardless of what algorithm is used, there is always the issue in a Hybrid architecture of when to use it. This is sometimes called the issue of interleaving reaction and planning.

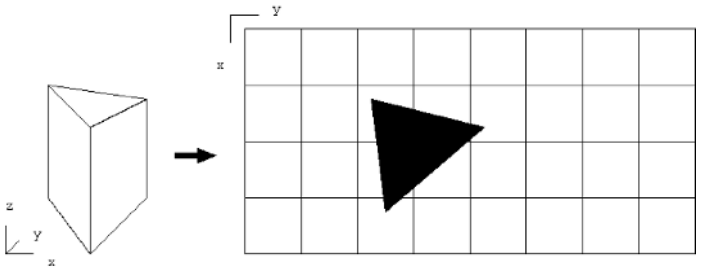


Figure 10.1 Reduction of a 6DOF world space to a 2DOF configuration space.

10.2 Configuration Space

CONFIGURATION SPACE

The physical space robots and obstacles exist in can be thought of as the world space. The *configuration space*, or *Cspace* for short, is a data structure which allows the robot to specify the position (location and orientation) of any objects and the robot.

A good Cspace representation reduces the number of dimensions that a planner has to contend with. Consider that it takes six dimensions (also called *degrees of freedom* or *DOF*) to represent precisely where an object is.

DEGREES OF FREEDOM

A person may specify the location of the object as a (x, y, z) coordinate in some frame of reference. But an object is three-dimensional; it has a front and back, top and bottom. Three more degrees are needed to represent where the front of the chair is facing, whether it is tilted or not, or even upside down. Those are the Euler (pronounced "Oiler") angles, $\phi, \theta,$, also known as pitch, yaw, and roll.

Six degrees of freedom is more than is needed for a mobile ground robot in most cases for planning a path. The z (height) coordinate can be eliminated if every object the robot sits on the floor. However, the z coordinate will be of interest if the robot is an aerial or underwater vehicle. Likewise, the Euler angles may be unnecessary. Who cares which way the robot is facing if all the robot wants to do is to plan a path around it? But the pitch of a planetary rover or slope of an upcoming hill may be critical to a mission over rocky terrain.

Fig. 10.1 shows a transformation of an object into Cspace. In general, metric path planning algorithms for mobile robots have assumed only two DOF, including for the robot. For path planning purposes, the robot can be modeled as round so that the orientation doesn't matter. This implicitly assumes

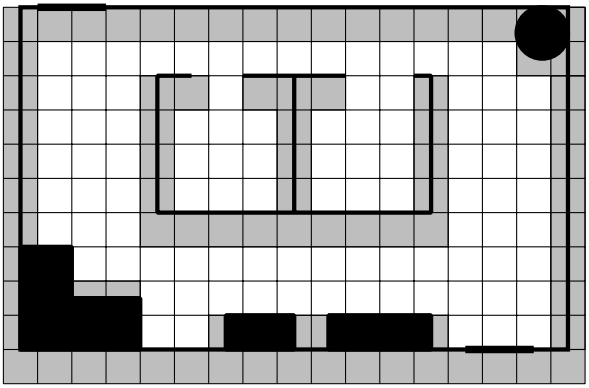


Figure 10.6 Regular grid.

fares. It should also be noted that the curved edges in a GVG do not matter to graph theory or graph algorithms. It is only the length, not the physical reality, of the edges that make any difference.

10.3.3 Regular grids

Another method of partitioning the world space is a regular grid. The regular grid method superimposes a 2D Cartesian grid on the world space, as shown in Fig. 10.6. If there is any object in the area contained by a grid element, that element is marked occupied. Hence, regular grids are often referred to as occupancy grids. Occupancy grids will be detailed in Ch. 11.

Regular grids are straightforward to apply. The center of each element in the grid can become a node, leading to a highly connected graph. Grids are either considered *4-connected* or *8-connected*, depending on whether they permit an arc to be drawn diagonally between nodes or not.

4-CONNECTED NEIGHBORS
8-CONNECTED NEIGHBORS
DIGITIZATION BIAS

Unfortunately, regular grids are not without problems. First, they introduce *digitization bias*, which means that if an object falls into even the smallest portion of a grid element, the whole element is marked occupied. This leads to wasted space and leads to very jagged objects. To reduce the wasted space, regular grids for an indoor room are often finely grained, on the order of 4- to 6-inches square. This fine granularity means a high storage cost, and a high number of nodes for a path planning algorithm to consider.

of how it is visualized, each node is evaluated to determine which is the most plausible move. The above figure shows what the algorithm "sees" at this point in the execution. The choices evaluate to:

$$\begin{aligned} f^*(B) &= g^*(B) + h^*(B) = 1 + 2.24 = 3.24 \\ f^*(D) &= g^*(D) + h^*(D) = 1.4 + 1.4 = 2.8 \end{aligned}$$

A path going from $A - D - ? - E$ has the potential to be shorter than a path going from $A - B - ? - E$. So, D is the most plausible node. Notice that A* can't eliminate a path through B because the algorithm can't "see" a path that actually goes from D to E and determine if it is indeed as short as possible.

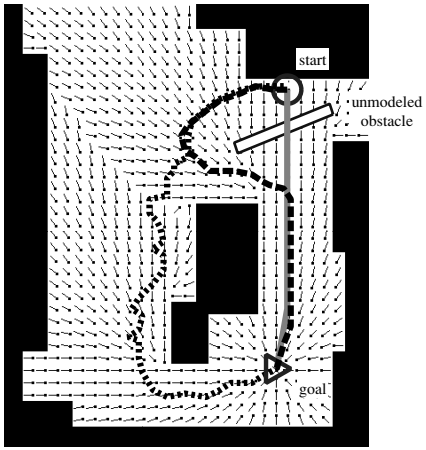
At step 2, A* recurses (repeats the evaluation) from D, since D is the most plausible, as shown in Fig. 10.10.

The two options from D are E and F, which are evaluated next:

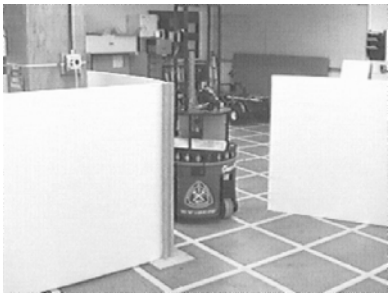
$$\begin{aligned} f^*(E) &= g^*(E) + h^*(E) = 2.8 + 0 = 2.8 \\ f^*(F) &= g^*(F) + h^*(F) = 2.4 + 1.0 = 3.4 \end{aligned}$$

Now the algorithm sees that E is the best choice among the leaves of the search tree, including the branch through B. (If B was the best, then the algorithm would have changed branches.) It is better than F and B. When it goes to expand E, it notices that E is the goal, so the algorithm is complete. The optimal path is $A - D - E$, and we didn't have to explicitly consider $A - B - F - E$. There are other ways to improve the procedure described so far. $f^*(F)$ didn't need to be computed if the algorithm looks at its choices and sees that one of them is the goal. Any other choice has to be longer because edges aren't allowed to be negative, so $D - F - E$ has to be longer than $D - E$.

Another important insight is that any path between A and E has to go through D, so the B branch of the search tree could have been pruned. Of course, in the above example the algorithm never had an opportunity to notice this because B was never expanded. It's easy to imagine in a larger graph that there might be a case where after several expansions through D, the leaf at B in the search tree came up the most plausible. Then the algorithm would have expanded A and seen the choices were D. Since D already occurred in another branch, with a cheaper $g^*(D)$, the B branch could be safely pruned.



a.



b.

Figure 10.13 Layout showing unmodeled obstacle. a.) Gray line shows expected path, long dashed line the actual path with Trulla, and short dashed line shows purely reactive path. b.) *Clementine* opportunistically turning.

Computing the optimal path from every location to the goal actually helps with reactive execution of the path. It means that if the robot can localize itself on the *waypoint* map, it can read the optimal subgoal for `move-to-goal` on each update. If the robot has to swing wide to avoid an unmodeled obstacle in Fig. 10.13, the robot automatically becomes redirected to the optimal path without having to replan. Note how the metric path becomes a virtual sensor, guiding the move-to-goal behavior replacing the direct sensor data. This is a rich mechanism for the deliberative and reactive components of Hybrid architectures to interact.

This approach eliminates subgoal obsession, since the robot can change "optimal" paths reactively and opportunistically move to a closer waypoint. As with most things in life, too much of a good thing is bad. At some point though, the sheer number of unmodeled obstacles might force the robot to get trapped or wander about, changing subgoals but making no real progress. The D* solution to this problem is to continuously update the map and dynamically repair the A* paths affected by the changes in the map. D* represents one extreme on the replanning scale: *continuous replanning*.

CONTINUOUS REPLANNING

Continuous replanning has two disadvantages. First, it may be too computationally expensive to be practical for a robot with an embedded processor

and memory limitations, such as a planetary rover. Second, continuous replanning is highly dependent on the sensing quality. If the robot senses an unmodeled obstacle at time T1, it computes a new path and makes a large course correction. If it no longer senses that obstacle at time T2 because the first reading was a phantom from sensor noise, it will recompute another large course correction. The result can be a robot which has a very jerky motion and actually takes longer to reach the goal.

EVENT-DRIVEN REPLANNING

In the cases of path planning with embedded processors and noisy sensors, it would be desirable to have some sort of *event-driven* scheme, where an event noticeable by the reactive system would trigger replanning. Trulla uses the dot-product of the intended path vector and the actual path vector. When the actual path deviates by 90° or more, the dot product of the path vector and the actual vector the robot is following becomes 0 or negative. Therefore the dot product acts as an affordance for triggering replanning. The robot doesn't have to know why it is drifting off-course, only that it has drifted noticeably off-course.

This is very good for situations that would interfere with making progress on the originally computed path. In effect, situations where the real world is less amenable to reaching the intended goal. But it doesn't handle the situation where the real world is actually friendlier. In Fig. 10.14, an obstacle thought to be there really isn't. The robot could achieve a significant savings in navigation by opportunistically going through the gap.

Such opportunism requires the robot to notice that the world is really more favorable than originally modeled. A continuous replanner such as D* has a distinct advantage, since it will automatically notice the change in the world and respond appropriately, whereas Trulla will not notice the favorable change because it won't lead to a path deviation. It is an open research question whether there are affordances for noticing favorable changes in the world that allow the robot to opportunistically optimize it path.

10.7 Summary

Metric path planning converts the world space to a configuration space, or *Cspace*, representation that facilitates path planning. Cspace representations such as generalized Voronoi diagrams exploit interesting geometric properties of the environment. These representations can then be converted to graphs, suitable for an A* search. Since Voronoi diagrams tend to produce sparser graphs, they work particularly well with A*. Regular grids work

approach presented here. In a Bayesian approach, the sensor model generates conditional probabilities of the form $P(s|H)$. These are then converted to $P(H|s)$ using Bayes' rule. Two probabilities, either from two different sensors sensing at the same time or from two different times, can be fused using Bayes' rule.

11.3.1 Conditional probabilities

PROBABILITY FUNCTION

To review, a *probability function* scores evidence on a scale of 0 to 1 as to whether a particular event **H** (**H** stands for "hypothesis") has occurred given an experiment. In the case of updating an occupancy grid with sonar readings, the experiment is sending the acoustic wave out and measuring the time of flight, and the outcome is the range reading reporting whether the region being sensed is Occupied or Empty.

Sonars can observe only one event: whether an element $grid[i][j]$ is Occupied or Empty. This can be written $\mathbf{H} = \{H, \neg H\}$ or $\mathbf{H} = \{Occupied, Empty\}$.

The probability that H has really occurred is represented by $P(H)$:

$$0 \leq P(H) \leq 1$$

An important property of probabilities is that the probability that H didn't happen, $P(\neg H)$, is known if $P(H)$ is known. This is expressed by:

$$1 - P(H) = P(\neg H)$$

As a result, if $P(H)$ is known, $P(\neg H)$ can be easily computed.

UNCONDITIONAL PROBABILITIES

Probabilities of the form $P(H)$ or $P(\neg H)$ are called *unconditional probabilities*. An example of an unconditional probability is a robot programmed to explore an area on Mars where 75% of the area is covered with rocks (obstacles). The robot knows in advance (or *a priori*) that the next region it scans has $P(H = Occupied) = 0.75$.

Unconditional probabilities are not particularly interesting because they only provide *a priori* information. That information does not take into account any sensor readings, S. It is more useful to a robot to have a function that computes the probability that a region $grid[i][j]$ is either Occupied or Empty given a particular sensor reading s. Probabilities of this type are called *conditional probabilities*. $P(H|s)$ is the probability that H has really occurred given a particular sensor reading s (the "|" denotes "given"). Unconditional probabilities also have the property that $P(H|s) + P(\neg H|s) = 1.0$.

CONDITIONAL PROBABILITIES

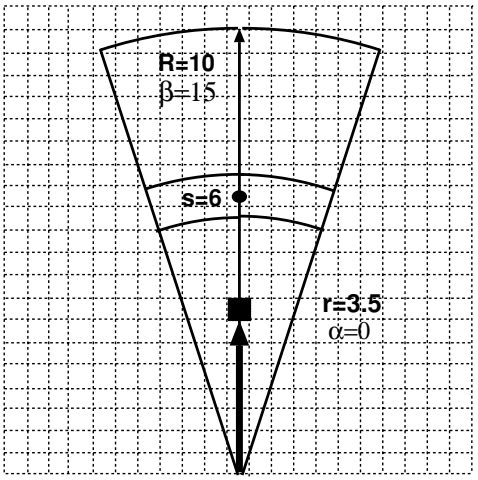


Figure 11.4 Example 1: Updating an element in Region I (sonar reading of 6).

Note that unlike an element in Region I, an element in Region II can have a probability of being empty of 1.0.

To see how these formulas would be applied, consider the example in Fig. 11.4. The sonar has returned a range reading of 6.0 feet with a tolerance of ± 0.5 feet. The $Max_{occupied}$ value is 0.98. The robot is shown on a grid, and all elements are measured relative to it. The element of interest $grid[i][j]$ is shown in black, and is at a distance $r = 3.5$ feet and an angle of $\alpha = 0°$ from the robot. In a computer program, r and α would be computed from the distance and arctangent between the element of interest and the element representing the origin of the sonar, but for the sake of focus, these examples will give r and α.

The first step is to determine which region covers the element. Since $3.5 < (6.0 - 0.5)$, the element is in Region II. Therefore, the correct formulas to apply are those in Eqn. 11.2:

$$\begin{array}{rcccccl} P(Empty) & = & \frac{(\frac{R-r}{R})+(\frac{-\alpha}{\beta})}{2} & = & \frac{(\frac{10-3.5}{10})+(\frac{15-0}{15})}{2} & = & 0.83 \\ P(Occupied) & = & 1.0 - P(Empty) & = & 1 - 0.83 & = & 0.17 \end{array}$$

The example in Fig. 11.5 shows an element in Region I. The probability for the element in black is computed the same way, only using the equations for that region.

11.6 Comparison of Methods

Occupancy grid methods have their unique advantages and disadvantages. Bayesian and Dempster-Shafer theory are formal theories, and other readings from other sensor modalities, such as range from stereo or a laser, can be easily fused as long as there is a sensor model. HIMM is limited to sonars but it has significant computational advantages. As seen in Fig. 11.14, all three produce similar occupancy grids, with a slight advantage going to Bayesian and Dempster-Shafer grids. In practice, Bayesian and Dempster-Shafer have fewer parameters to tune, making them more straightforward to adapt to new environments.

11.6.1 Example computations

The similarities and differences between the three methods is best seen by an example. The following example covers how to initialize the occupancy grid, compute the score at a grid element for a sensor reading, update the grid, and repeat for three different observations.

Step 1: Initialize the Occupancy Grid.

Consider a robot beginning to map a new area. The occupancy grid shown in Fig. 11.15 covers an area of 12 units by 10 units. The grid is an array of size 24 x 21, with 2 grid elements per unit of distance. The grid starts in an initial unsensed state. In a Bayesian approach, each element in the grid would be a structure `P` with two fields: $P(Occupied)$ and $P(Empty)$. The value of each field depends on the unconditional probability that the area represented by the grid is occupied or empty. Unless there is some prior knowledge, the assumption is that an element has equal chances of being occupied or empty. This translates to $P(Occupied) = P(Empty) = 0.5$. Every element in the grid would start with (0.5, 0.5). In a Dempster-Shafer implementation, each element in the grid would be a structure `Bel` with three fields: $m(Occupied), m(Empty)$ and $m(dontknow)$. Since the grid represents areas that have not been sensed, the entire belief mass m is initialized as $m(dontknow) = 1.0$. Every element in the grid would start with (0.0, 0.0, 1.0). Every element in a HIMM occupancy grid would be a single 8-bit integer, and would be initialized to 0.

Consider how three different sonar updates create a certainty value for a particular grid element, $grid[3][10]$, shown in Fig. 11.15. At time t_1, the sonar

Step 2: Compute the uncertainty of the observation.
The second step is to compute the uncertainty score of an observation, remembering that a grid element will only have a score if it is covered by the sonar reading. This computation can be done in a series of sub-steps. The process begins by determining whether $grid[3][10]$ falls in Region I or Region II, since this specifies which equations or increment to use. Region I, $Occupied$, extends for $s \pm tolerance$. The test for falling in Region I is the same for all three methods: if r satisfies $s - tolerance \leq r \leq s + tolerance$, then it is in Region I. In this case, $s = 9$, $tolerance = 0.5$, and $r = 9$, and the substitution results in $9 - 0 \leq 9 \leq 9 + 0.5$ being true. Therefore $grid[3][10]$ is in Region I.

At this point, the three methods diverge in computing the "score" from the reading at t_1. The next step in Bayesian methods is to compute the probability, $P(s|Occupied)$, that the sensor s will correctly report that $grid[3][10]$ is $Occupied$ if there is really something at $s = 9$. This is done using Eqn. 11.1:

$$\begin{aligned} P(s|Occupied) &= \frac{(\frac{R-r}{R}) + (\frac{-\alpha}{2})}{2} \times Max_{occupied} \\ &= \frac{(\frac{10-9}{10}) + (\frac{15-0}{15})}{2} \times 0.98 = 0.54 \\ P(s|Empty) &= 1.0 - P(s|Occupied) \\ &= 1.0 - 0.54 = 0.46 \end{aligned}$$

Dempster-Shafer theory uses Eqn. 11.8, which produces essentially the same score for the sensor reading as with the Bayesian method:

$$\begin{aligned} m(Occupied) &= \frac{(\frac{R-r}{R}) + (\frac{-\alpha}{2})}{2} \times Max_{occupied} \\ &= \frac{(\frac{10-9}{10}) + (\frac{15-0}{15})}{2} \times 0.98 = 0.54 \\ m(Empty) &= 0.0 \\ m(dontknow) &= 1.00 - m(Occupied) \\ &= 1.0 - 0.54 = 0.46 \end{aligned}$$

The HIMM score is the I term in Eqn. 11.11. Since $grid[3][10]$ is in Region 1 of the HIMM sonar model, $I = I^+ = +3$.

The HIMM updating scheme is simple where:

$$\begin{aligned} grid[3][10] &= grid[3][10] + 3 \\ &= 3 + 3 = 6 \end{aligned}$$

At t_3, the sonar returns a value of 8.5 units. The robot has moved to the side and rotated; it is now 6.7 units from the grid element with an α of $5°$. In this case $grid[3][10]$ is in Region II for the Bayesian and Dempster-Shafer sonar models, and is not affected by the HIMM model at all.

The probability score for the Bayesian model is computed using Eqn. 11.2 instead of Eqn. 11.1:

$$\begin{aligned} P(s|Empty) &= \frac{(\frac{R-r}{R}) + (\frac{-\alpha}{\beta})}{2} \\ &= \frac{(\frac{10-6.7}{10}) + (\frac{15-5}{15})}{2} = 0.50 \\ P(s|Occupied) &= 1.0 - P(s|Empty) = 1.0 - 0.50 = 0.50 \end{aligned}$$

The result happens to be an almost even probability that $grid[3][10]$ is occupied. This probability is then substituted into Bayes rule (Eqn. 11.6) with the previously stored probability:

$$\begin{aligned} P(O|s_{t_3}) &= \frac{P(s_{t_3}|O)P(O|s_{t_0})}{P(s_{t_3}|O)P(O|s_{t_0}) + P(s_{t_1}|E)P(E|s_{t_0})} \\ &= \frac{(0.50)(0.72)}{(0.50)(0.72) + (0.50)(0.28)} \\ &= 0.72 \\ P(E|s_{t_3}) &= 1 - P(O|s_{t_3}) = 0.28 \end{aligned}$$

The Dempster-Shafer belief function is computed using Eqn. 11.9, yielding $m(Occupied) = 0.0, m(Empty) = 0.50), m(dontknow) = 0.50)$. The difference between the probability and belief function is that the $\neg Empty$ score was assigned to $P(s|Occupied)$ in the Bayesian method and to $m(dontknow)$ in the Dempster-Shafer. The combination is shown in Fig. 11.16c, and produces:

$$m(Occupied) = \frac{(0.86)(0.5)}{1.0 - (0.86)(0.5)} = 0.76$$

$$m(dontknow) = \frac{(0.14)(0.5)}{1.0 - (0.86)(0.5)} = 0.12$$

$$m(Empty) = \frac{(0.14)(0.5)}{1.0 - (0.86)(0.5)} = 0.12$$

Since $grid[3][10]$ is not affected by the HIMM sonar model for the reading at t_3, there is no update.

The above computations can be summarized as follows. The score for $grid[3][10]$ at each observation is:

sonar certainty:	Bayesian		Dempster-Shafer			HIMM
	$P(s\|O)$	$P(s\|E)$	$m(O)$	$m(E)$	m($dontknow$)	
t_1	0.54	0.46	0.56	0.00	0.46	+3
t_2	0.69	0.31	0.69	0.00	0.31	+3
t_3	0.50	0.50	0.00	0.50	0.50	n/a

Notice the differences in the Bayesian and Dempster-Shafer scores. The numbers are the same, but where those numbers go is quite different. At t_2, both methods score the occupancy of the grid element as 0.69. But the Bayesian scores the emptiness as 0.31, while Dempster-Shafer doesn't commit to the area being empty; rather it can't tell if it is empty or occupied. At t_3, there is no HIMM score because $grid[3][10]$ is not covered by the HIMM sonar model's field of view.

The updated value of $grid[3][10]$ after each observation, that is, the combination of the current score with the previous score, is:

after update:	Bayesian		Dempster-Shafer			HIMM
	$P(O\|s)$	$P(E\|s)$	$m(O)$	$m(E)$	m($dontknow$)	
t_1	0.54	0.46	0.54	0.00	0.46	3
t_2	0.72	0.28	0.86	0.00	0.14	6
t_3	0.72	0.28	0.76	0.12	0.12	6

Notice that the end results of the Bayesian and Dempster-Shafer fusion methods are very similar, though the intermediate values are different. In the HIMM, the value of $grid[3][10]$ after t_3 is 6 because nothing is done to it after t_2; it is neither incremented nor decremented.

11.6.2 Performance

Fig. 11.14 shows the three methods used to generate occupancy grids for data collected from the same hallway. Performance scores are easy to compute. The ground truth is expressed as a "perfect" occupancy grid, manually

or crosstalk, obstacles might be missed or appear at the wrong distance. If the robot is stationary, HIMM will generate high belief in an incorrect map. Bayesian and Dempster-Shafer theory also suffer from the same defect. Since they usually cover a larger area, the problem with gaps in walls is usually avoided. But problems with phantom readings still cause incorrect maps.

The plots of the rate of accrual of belief show that multiple identical readings will cause the robot to quickly believe that its occupancy grid is correct. Once $P(H|S)$ or $m(H)$ reach 1.0, there is no revision downward. HIMM can revise belief because it subtracts strictly based on the current reading. But HIMM must have a new, contradictory reading to cause this to happen.

The reason Bayesian and Dempster-Shafer methods degenerate when the robot is stationary and receives multiple, identical readings is because the assumption that the observations are independent has been violated. If the robot is at the same location sensing the same object, the value of reading $S_{t_{n+1}}$ is likely to be the same as S_{t_n}. Since the robot has not moved, the observations cannot be considered to be taken from two different experiments or by two different observers. This serves as a cautionary note about making simplifying assumptions: it is important to understand when those assumptions lead to counterproductive results.

11.6.4 Tuning

Fig. 11.12 shows the performance of the three updating methods for a hallway with significant specular reflection. All three methods show the hallway as being wider than it really is. This would be a serious problem for navigation and obstacle avoidance. The sensor noise was not eliminated by the use of an occupancy grid. In many cases, a large amount of sensor noise can be eliminated by tuning the model and updating algorithms.

Therefore an important criterion for an algorithm is how easily it can be tuned for a particular environment. For example, in environments which provoke a high degree of specular reflection in sonars, a $< 8°$ is often used to reduce the registration of noise in the occupancy grid. Why put false readings into the grid over a large area that will take several contradictory readings to eliminate? It can often take one or more days to tune a set of sonars which were producing near perfect occupancy grids in a laboratory for a new building.

3 WAYS TO TUNE

Occupancy grids can be tuned for a task environment in at least three ways. One way is to leave all the algorithms the same but concentrate on adjusting the physical equipment. For example, the time of flight of sound

readings will update the occupancy grid, reducing the amount and location of unknown areas, and creating a new goal. Hughes and Murphy[105] showed that this move-to-unknown-area behavior was suitable for indoor exploration and even localization. While the above behavior-oriented approaches are simple and easy to implement, they often are inefficient, especially when presented with two or more unexplored areas. Suppose a robot has encountered a hallway intersection; how does it choose which area to explore?

Two basic styles of exploration methods have emerged which rank unexplored areas and make rational choices: *frontier-based* and *generalized Voronoi graph* methods. Both work well for indoor environments; it is less clear how these work over large open spaces. Both use behaviors for navigation, but are different in how they set the navigational goals. This section provides a highly simplified overview of each method.

11.8.1 Frontier-based exploration

Frontier-based exploration was pioneered by Brian Yamauchi.[125] The approach assumes the robot is using a Bayesian occupancy grid (a Dempster-Shafer grid can be used as well). As shown in Fig. 11.22, when a robot enters a new area, there is a boundary between each area that has been sensed and is open and the area that has not been sensed. (The boundary between occupied areas and unknown areas are not interesting because the robot cannot go through the occupied area to sense what is behind it.) There are two such boundaries in Fig. 11.22; each of these lines form a *frontier* that should be explored.

FRONTIER

The choice of which frontier to be explored first can be made in a variety of ways. A simple strategy is to explore the nearest frontier first. Another is to explore the biggest frontier first. Since the world is unknown, the robot has no way of knowing if upon reaching a big frontier it will discover a wall just a meter away. This means that the robot might move across a room, briefly explore one area, then return back to almost at its starting point, explore that area, and then go to another place, and so on. In practice, this doesn't happen that often with indoor environments.

The size of the frontier can be measured by the number of edge cells. Every cell in the occupancy grid that the boundary runs through is considered an edge. If an edge "touches" an edge in one of its eight surrounding neighbors, the edges are connected and form the line. In order to eliminate the effects of

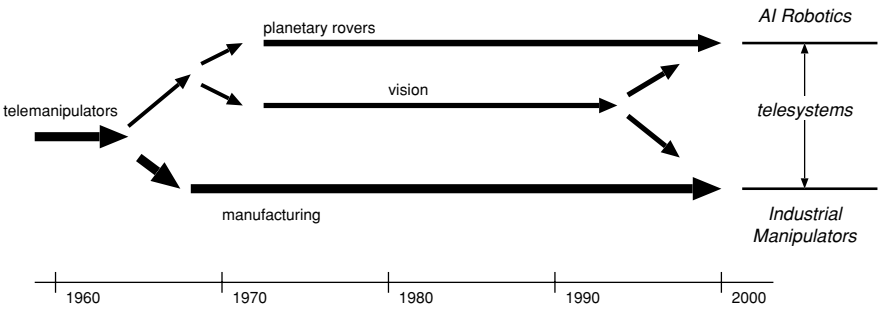


Figure 12.1 A timeline showing forks in development of robots.

vision and robotics was Avi Kak at Purdue University. His Cybermotion robot was one of the first to navigate in hallways using vision; in this case, a technique known as a Hough (pronounced "huff") transform.[80] In the early 1990's, with slow hardware, the robot could go 8 to 9 meters per second. The computer vision community went through the equivalent of a Reactive movement, moving to approaches now called *animate vision*, *purposive vision*, or *active vision*.

ANIMATE, PURPOSIVE, ACTIVE VISION

Another positive influence to reunite robotics and vision has been the various DARPA and ESPIRIT projects in mobility. Both agencies have provided funding for large-scale projects, such as fully autonomous off-road and highway vehicles. The size of the projects require hundreds of researchers from many universities to collaborate, providing the opportunity to cross-fertilize the fields.

A more cost-effective motivation to have roboticists work with vision, and vice versa, has been the various robot competitions. The prestigious AAAI Mobile Robot Competitions have changed the events and the rules each year to reward researchers who use computer vision in creative ways, instead of relying on the standard sonars. A newer event, RoboCup, mandates vision. There is no other way to play soccer than to see the ball and the players in real-time. The competitions have already spawned at least one commercial product: the Cognachrome fast vision board developed by Newton Labs. The ground and aerial robot competitions sponsored by the Association for Unmanned Vehicle Systems also promote the integration of computer vision with robotics, but winning those competitions still largely depends on hardware, not software algorithms.

Bibliography

[1] Albus, J., and Proctor, F.G., "A Reference Model Architecture for Intelligent Hybrid Control Systems," proceedings of the *International Federation of Automatic Control*, San Francisco, CA, June 30–July 5, 1996.

[2] Allocca, J. A., and Stuart, A., *Transducers: Theory and Application*, Prentice-Hall, 1984.

[3] Anderson, M.O., McKay, M.D., Richardson, B.A., "Multirobot Automated Indoor Floor Characterization Team," proceedings of the *1996 IEEE International Conference on Robotics and Automation*, Minneapolis, MN, April, 1996, pp. 1750–1753.

[4] Anderson, T.L., and Donath, M., "Animal Behavior as a Paradigm for Developing Robot Autonomy," *Robotics and Autonomous Systems*, vol. 6, 1990, pp. 145–186.

[5] Arbib, M., "Perceptual Structures and Distributed Motor Control," *Handbook of Physiology—The Nervous System II*, ed. Brooks, pp. 1449–1465, 1981.

[6] Arbib, M., "Schema Theory," *The Handbook of Brain Theory and Neural Networks*, Michael A. Arbib, ed., MIT Press, pp. 830–834.

[7] Arbib, M., and Liaw, J.-S., "Sensorimotor transformations in the worlds of frogs and toads," *Computational Theories of Interaction and Agency*, Philip Agre and Stanley J. Rosenschein, ed., MIT Press, 1996, pp. 53–80.

[8] Arkin, R.C., "Towards Cosmopolitan Robots: Intelligent Navigation in Extended Man-Made Environments," COINS Technical Report 87-80, *Computer and Information Science*, University of Massachusetts at Amherst, 1987.

[9] Arkin, R., personal communication, 1988.

[10] Arkin, R., *Behavior-Based Robotics*, MIT Press, 1998.

[11] Arkin, R., and MacKenzie, D., "Temporal Coordination of Perceptual Algorithms for Mobile Robot Navigation," *IEEE Transactions on Robotics and Automation*, vol. 10, no. 3, June, 1994, pp. 276–286.

[12] Arkin, R.C. and Murphy, R.R., "Autonomous Navigation in a Manufacturing Environment," *IEEE Transactions on Robotics and Automation*, vol. 6, no. 4, August, 1990, pp. 445–454.

[92] McKenzie, D.C.; Arkin, R.C.; Cameron, J.M., " Multiagent mission specification and execution," *Autonomous Robots*, vol.4, no.1, 1997, p. 29–52.

[93] Meystel, A., "Knowledge Based Nested Hierarchical Control," in *Advances in Automation and Robotics*, vol. 2, Ed. G. Saridis, JAI Press, Greenwich, CT, 1990, pp. 63–152.

[94] Moravec, H., *Mind Children*, Harvard University Press, 1988.

[95] Moravec, H.P., "Sensor Fusion in Certainty Grids for Mobile Robots," *AI Magazine*, vol. 9, no. 2, Summer, 1988, pp. 61–74.

[96] Moravec, H., *Robot: Mere Machine to Transcendent Mind*, Oxford Press, 1998.

[97] Murphy, R.R., "A Strategy for the Fine Positioning of a Mobile Robot using Texture," proceedings of *SPIE Mobile Robots IV*, Philadephia, PA, Nov. 5-10, 1989, pp. 267–279.

[98] Murphy, R. R., "An Artificial Intelligence Approach to the 1994 AUVS Unmanned Ground Vehicle Competition," *1995 IEEE International Conference on Systems, Man and Cybernetics*, Oct. 1995, Vancouver, BC., pp. 1723–1728.

[99] Murphy, R.R., "Biological and Cognitive Foundations of Intelligent Sensor Fusion," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 26, No. 1, Jan, 1996, pp. 42–51.

[100] Murphy, R.R., "Use of Scripts for Coordinating Perception and Action," *IROS-96*, Nov. 1996, Osaka, Japan, pp. 156–161.

[101] Murphy, R. 1997 International Ground Robotics Vehicle Competition," Robotics Competition Corner, *Robostics and Autonomous Systems*, Elsevier, vol. 20, 1997.

[102] Murphy, R., "Dempster-Shafer Theory for Sensor Fusion in Autonomous Mobile Robots," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 2, April, 1998.

[103] Murphy, R.R., Gomes, K., and Hershberger, D., "Ultrasonic Data Fusion as a Function of Robot Velocity," *1996 SPIE Sensor Fusion and Distributed Robotic Agents*, Nov. 1996, Boston, MA., pp. 114–126.

[104] Murphy, R. R., Hawkins, D.K., and Schoppers, M.J., "Reactive Combination of Belief Over Time Using Direct Perception," *IJCAI-97*, Nagoya, Japan, Aug. 1997, pp. 1353–1358.

[105] Murphy, R.R., and Hughes, K., "Ultrasonic Robot Localization using Dempster-Shafer Theory," *SPIE Stochastic Methods in Signal Processing, Image Processing, and Computer Vision*, invited session on Applications for Vision and Robotics, San Diego, CA, July 19-24, 1992.

[106] Murphy, R., Hughes, K., Noll, E., and Marzilli, A., "Integrating Explicit Path Planning with Reactive Control for Mobile Robots using Trulla," *Robotics and Autonomous Systems*, no. 27, 1999, pp. 225–245.

[124] Schöne, H., *Spatial Orientation: The Spatial Control of Behavior in Animals and Man*, translated by C. Strausfeld, Princeton University Press, NJ, 1984.

[125] Schultz, A., Adams, W., and Yamauchi, B., "Integrating Exploration, Localization, Navigation and Planning With a Common Representation," *Autonomous Robots*, vol. 6 no. 3, May 1999, pp. 293–308.

[126] Shafer, G., *A Mathematical Theory of Evidence*, Princeton University Press, 1976.

[127] Shaffer, G., Gonzalez, J., Stentz, A., "Comparison of two range-based pose estimators for a mobile robot," *SPIE Mobile Robots VI*, Boston, MA, USA; Nov. 1992, pp. 661–7.

[128] Sheridan, T., "Space Teleoperation Through Time Delay: Review and Prognosis," *IEEE Transactions on Robotics and Automation*, vol. 9, no. 5, Oct. 1993, pp. 592–605.

[129] Simmons, R., *AI Magazine*, Summer, vol. 16, no. 2, 1995, pp. 19–30.

[130] Simmons, R., Fernandez, J., Goodwin, R., Koenig, S., O'Sullivan, J., "Xavier: An Autonomous Mobile Robot on the Web," *Robotics and Automation Society Magazine*, June, 1999.

[131] Simmons, R., Goodwin, R., Haigh, K., Koenig, S., and O'Sullivan, J., "A Layered Architecture for Office Delivery Robots," proceedings *Autonomous Agents 97*, 1997, pp. 245–252.

[132] Slack, M., "Navigation templates: mediating qualitative guidance and quantitative control in mobile robots," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 23, no. 2, 1993, p. 452–66.

[133] Smith, R., and Cheeseman, P., "On the Representation of and Estimation of Spatial Uncertainty," *International Journal of Robotics Research*, vol. 5, 1986, pp. 56–68.

[134] "Sojourner's 'Smarts' Reflect Latest in Automation," JPL Press Release, Aug. 8, 1997.

[135] Stark, L., and Bowyer, K., *Generic Object Recognition Using Form and Function*, World Scientific, 1996.

[136] Stentz, A., "The Focussed D* Algorithm for Real-Time Replanning," proceedings *1995 International Joint Conference on Artificial Intelligence*,Montreal, CA, Aug. 1995, pp. 1652–1659.

[137] Swain, M., and Ballard, D., "Color indexing," *International Journal of Computer Vision*, vol. 7, 1991, pp. 11–32.

[138] Thorpe, C.E., "Path Relaxation: Path Planning for a Mobile Robot," *1984 International Conference on AI (AAAI-84)*, 1984, pp. 318–321.

[139] Thrun,S., Bennewitz, M., Burgard, M., Dellaert, F., Fox, D., Haehnel, C., Rosenberg, C., Roy, N., Schulte, J., and Schulz, D., "MINERVA: A second generation mobile tour-guide robot," proceedings *1999 IEEE International Conference on Robotics and Automation*, vol. 3, pp. 1999–2005.