

Designing Behavior-based Robots

Jonathan H. Connell
Johuco Ltd., Box 385, Vernon CT 06066
<http://www.johuco.com>

In this paper, we describe a layered parallel control strategy developed at MIT. We will discuss the design of different behavioral layers and techniques for coordinating them to achieve tasks. These principles will be illustrated by an examination of several existing, large-scale mobile robots.

A LAYERED ARCHITECTURE

Building complete systems that intelligently connect sensors to actuators is a challenging endeavor. We believe a lot of the difficulty stems from the "traditional" artificial intelligence approach of breaking a control system into a number of monolithic functional slices (see Figure 1a). There is typically some perception system which is then followed by a modelling component. The output of this then feeds a general-purpose activity planner whose directives are carried out by an execution monitoring stage. The problem with having just one perception component is that it has to be complete enough to provide all the information that might be needed by succeeding stages. Also such large, comprehensive systems are notoriously slow – not a good feature for a real-time robot. Making even one such system is itself a large undertaking requiring teams of researchers many years to develop. Because these efforts are typically carried in isolation from each other, there is no guarantee that they will have compatible interfaces in the end. The perceptual system may segment the world according to edges and textures patches, whereas the planner might want to deal with individuated "objects". The planner might also presume that properties, such as the type of material composing an object, will be provided even though they may not be derivable from the available sensor data.

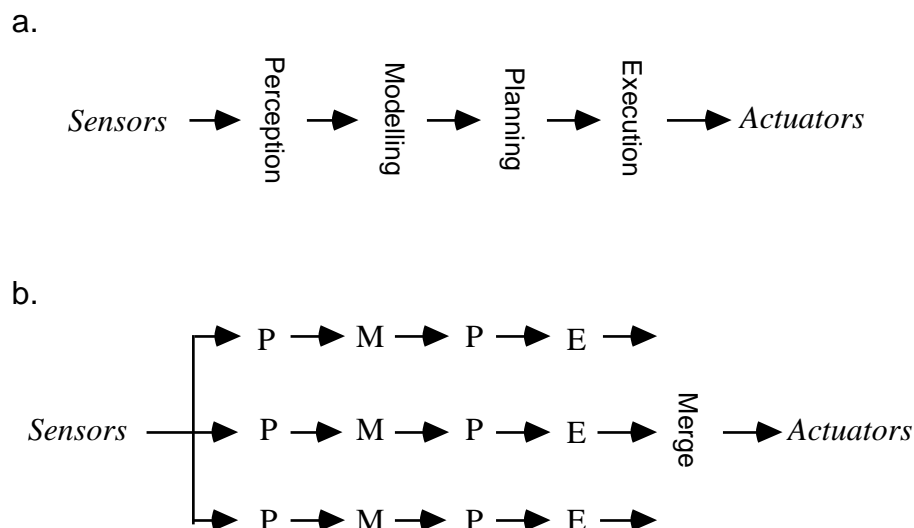


Figure 1 - Two types of decomposition. a. The traditional chain of highly competent functional subsystems. b. Brooks's collection of parallel special-purpose control paths.

The Subsumption Architecture devised by Brooks at MIT provides an alternative to this type of system. Instead of having a single chain of general-purpose functional slices, there are a number of parallel control paths each with its own perception, modelling, planning, and execution component (see Figure 1b). These paths are typically associated with some task, such as following walls, or grasping objects. The advantage of such a system is that each component only has to be competent enough to support the function delegated to its layer of control. Such special-purpose modules are much easier to build than their more complete counterparts, and typically run quite rapidly. In addition, the competence of a Subsumption Architecture system can be naturally expanded by simply adding more control paths to the existing system. The Subsumption Architecture has been successfully used in this manner on a number of robots.

Partial Representations

What the Subsumption Architecture does not specify is how to decompose a task into a number of separate behaviors. In practice, perception is by far the hardest problem in mobile robotics. With the limited sensory suite commonly available, it is difficult to extract semantically meaningful information from the world. Yet extracting such information in a timely manner is essential for the creature to behave intelligently. Thus, the form and content of those sensory representations which are easily computable from the data at hand is a key factor in determining what sort of task-directed subsystems can be constructed.

In robotics, the standard approach has been to first try to reconstruct an accurate internal representation of the world, sort of a diorama. The system then measures and compares various aspects of this representation to make control decisions. However, building and using such a representation is fraught with a number of difficulties. Sensors must be properly calibrated and transformed into a global coordinate frame, then the information from different sensory modalities must be combined and integrated over time. Interpretation of the raw data to yield compatible types of information from different kinds of sensors is difficult. Maintaining the consistency of the representation across large spatial distance or over extended intervals of time is problematic. Accurate models also imply accurate control. Thus, some precise and properly compensated motion controller must be used to carefully execute the steps of the high-level plan while also handling any contingencies that may arise.

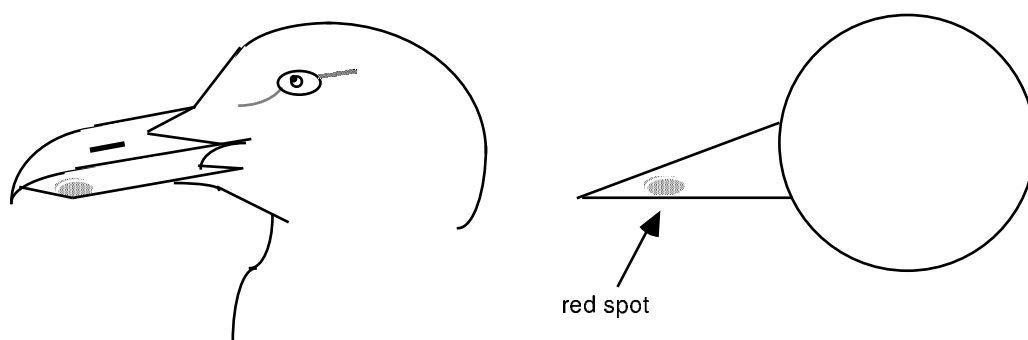


Figure 2 - Animals seem to use incomplete models for many activities. Baby seagulls respond just as well to the mockup on the right as they do to their own parent (left). The critical features are that the object must be pointed and have a red spot.

As a contrast to the usual robotics approach, let us examine some work from the field of ethology, the study of animal behavior. Much effort has been devoted to finding the “releasing” stimulus for particular behavioral pattern. By carefully controlled studies, researchers have been able to determine exactly which features of a situation an animal is paying attention to. Typically, creatures do not have very detailed models of the objects they interact with. For instance, when baby seagulls detect the arrival of one of the parents, they raise their

heads, open their mouths, and start squeaking in a plea for food. The baby birds do not recognize their parents as individuals, nor are they good at distinguishing seagulls from other animals or even inanimate objects. The birds respond just as well to a simple mockup as to the real parent (Figure 2). The important condition seems to be the presence of a pointed object with a red spot near its tip. In their natural environment, this model works fine because the real parents are the only objects which fit the bill. The same sort of minimal representation has been discovered for many other animals as well. This suggests that we might be able to build reasonably competent mobile robots without investing a large amount of effort into building detailed representations.

The particular type of partial representation we have adopted is mostly local and mostly stateless. Our representations are essentially “snapshots” of certain key types of situations. In this respect, our work can be considered an instance of the “matched filter” approach that seems to pervade insect nervous systems. The idea is to recognize the class of environmental conditions which call for either a particular action or for a transition between modes of operation. The resulting situation-action rules (which we call “behaviors”) are then arranged into prioritized sets to control the robot. Since we use mostly “reflex-like” direct responses, the modelling, planning, and execution components of our control paths are almost non-existent.

Structuring Principles

If we have only minimal planning in each behavior, how can the robot's individual reflexes be coordinated to achieve some goal? We start by enumerating a sequence of local environmental configurations that would be experienced as the robot performs a specific task. We then devise a configuration of sensors such that each situation generates a uniquely recognizable signature. Finally, we build simple interpretation routines that link the detection of each situation to an appropriate primitive action or simple control law. Obviously, given our coarse world modelling, some of these stimulus-response pairs might be simultaneously activated. To overcome this, we impose a priority ordering on the whole set of reflexes. More specific rules generally take precedence over vaguer suggestions or default behaviors. Similarly, behaviors likely to be encountered only in the later phases of some task are ranked more highly than the behaviors used in the earlier stages of the task. Finally, tactical behaviors, which need to respond quickly as events change, take precedence over strategic behaviors embodying the robot's longer-term policy.

Strangely enough, the priority structure we impose can also play a representational role. In many cases, the elements of a situation recognized by some behavior are not unique; there may be several different environmental configurations with the same signature. Like a baby seagull, our robots rely on a sense of normalcy, expecting some statistical regularity in their experiences. It is the job of previous behaviors in a sequence to constrain circumstances to such an extent that this assumption is seldom violated. Sometimes the behavior itself may help further test its own triggering condition by slowly changing the relation of the robot to the external world. In such a case, the robot proceeds as if the situation is the desired one until some anomaly is detected. This detection is the responsibility of a number of other behaviors which look for counter-indicative aspects of the current situation. If such non-nominal features are found, these behaviors usurp control and help the robot recover from its mistake or redirect its attention elsewhere.

Still, how can a collection of such mostly local and mostly stateless routines avoid getting stuck in local minima or infinite loops? Our solution is to have the robot continuously monitor a number of global progress indicators derived from incoming sensory data. In our systems, there are typically a number of behaviors which check to see whether the robot is no longer advancing toward the goal or has failed to make substantial headway over an extended interval of time. When such problems occur, these special-purpose routines kick-in to unwedge or reinitialize the robot.

Using the above guidelines, behaviors can be chained together to generate “trajectories” for the robot. Typically, the robot starts by moving in some default direction to get out of its initially ambiguous situation. Eventually, it gets to a place where there is enough information for some more specific behavior to take over. This behavior then retains control until it has sufficiently changed the current situation as to make it recognizable to one of the follow-up behaviors. This repeated transfer of control continues until the robot reaches its goal state or fails to

meet some global progress measure. At this point, a different group of control behaviors is usually selected by setting some longer term state bit. The major advantage of specifying the robot's trajectory as a set of response rules, such as this, is that we avoid precommitting to a particular order of events. Thus, the robot can handle "unexpected" occurrences and can take advantage of any "short-cuts" that may appear.

THE CO-DESIGN OF SENSORS AND RULES

Let us now look at how to design good physical arrangements of sensors and interpret them appropriately to accomplish some task. The robot in this section, "TJ", was built on a 12" circular base (see figure 3) which is small enough to easily fit through doors, but large enough to carry most of the processing that is required. This is important as TJ was designed to act as a high-speed courier within an office building. In terms of control, the strategic part of navigation – where to go next – is handled by a standard symbolic program while the tactical part – moment-to-moment steering – is handled by a collection of reactive behaviors. To do its job, the symbolic layer maintains a coarse geometric map of the robot's world consisting of a number of landmarks and a number of paths between them. However, these paths have very little information other than length associated with them. Thus, the symbolic system must trust the competence of the behavior-based system to actually get it from one end to the other.

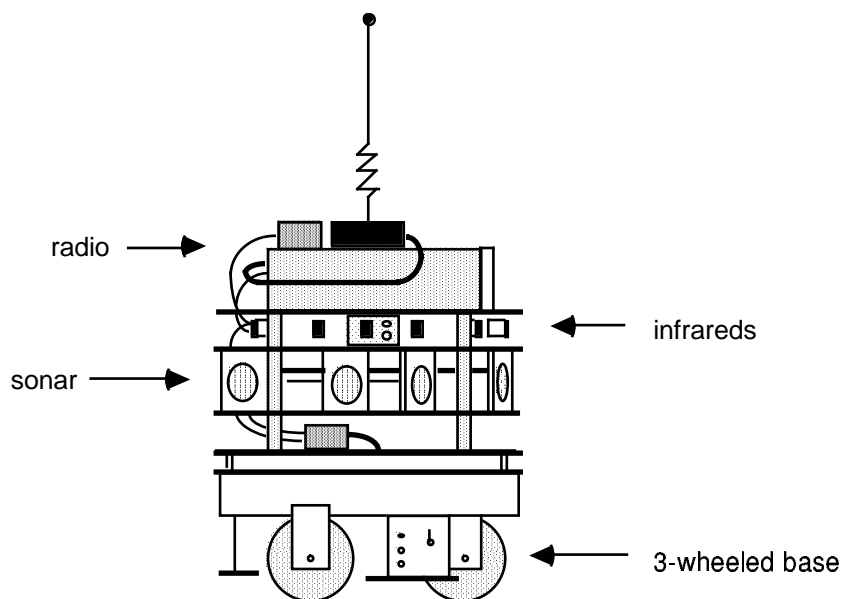


Figure 3 - TJ has a 3-wheeled omni-directional base and uses both sonar ranging and infrared proximity detection for navigation. An on-board network of microprocessors runs all local navigation behaviors.

One of the primary heuristics for successfully traversing a path segment is to follow along a wall. Figure 4 illustrates how we derived the control rules to accomplish this. First, we installed a long range (about 7 feet) infrared-based proximity detector (W) on the side of the robot to directly detect whether an obstacle (presumably a wall) is present anywhere near the robot. When a wall is present, we generally assume that the robot needs to be closer to it (a) and so we command the robot to gradually turn in the direction of the wall in this situation. To detect when the robot has gotten too close to the wall, we installed another shorter range (about 10 inches) IR sensor (M). When the robot perceives this situation (b), it steers gradually away from the wall.

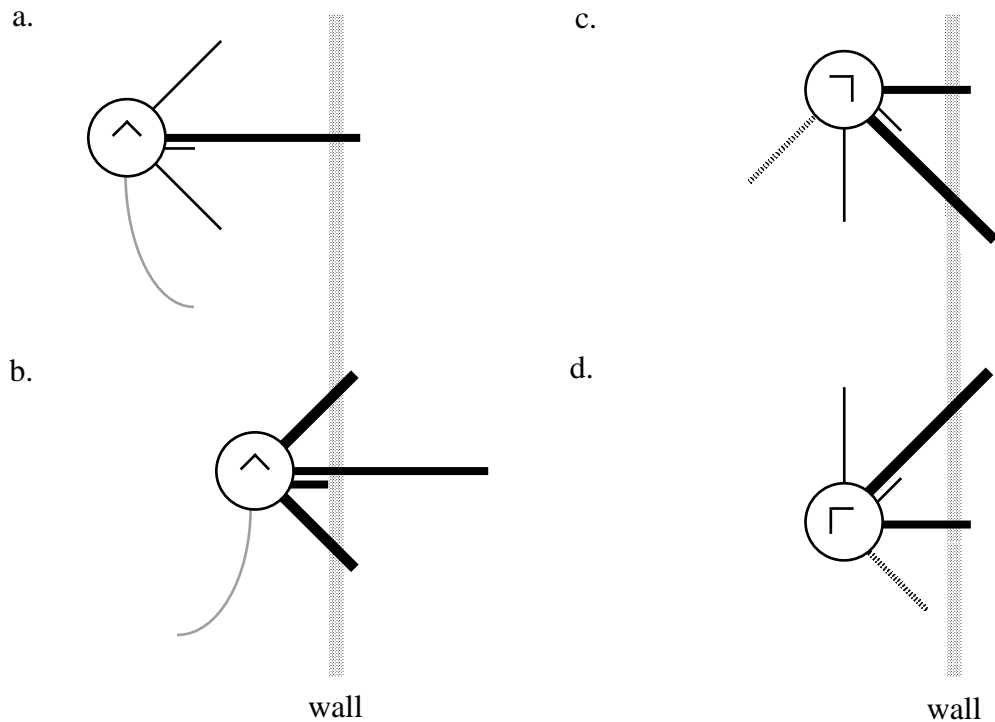


Figure 4 - Wall following depends on a number of “matched filters” that recognize particular situations. The overhead diagrams here show the sensor response obtained in several important classes of poses. A dark (versus light) line indicates that the corresponding IR sees an obstruction. The robot travels in the direction of the internal arrowhead; the dotted lines show its previous path.

We can write these two responses as an ordered list. In this list, the behaviors that come earlier can be overridden by ones further down. Thus, even though the long range wall IR is on when the robot is very close to the wall, rule number 2, based on the short middle sensor, takes over control of the robot. When neither of these rules is active, the robot goes straight as a default.

- 1: W → R
- 2: M → L

There are also two behaviors that adjust the angle of the robot with respect to the wall. To measure TJ's relative orientation we installed two more IR sensors (F and B) with a medium range (about 16 inches). If the robot is angled too steeply, the front diagonal IR will be on while the rear diagonal is off (c). To prevent the robot from getting too close, we create a behavior which causes the robot to turn away in this situation. Sometimes, however, the robot is misaligned in the other direction and will eventually leave the wall behind. In this situation the rear diagonal IR will see something while the front diagonal does not (d). To correct this, we install another behavior which causes the robot to veer back toward the wall. The final list of right-wall following control rules for the robot is then:

- 1: W → R
- 2: ¬F & B → R
- 3: M → L
- 4: F & ¬B → L

We assume that it is better to lose the wall (turn left) than to ram into it (by turning right). Thus, behaviors which turn the robot away from the wall take precedence over those guiding it closer. Also, since the act of adjusting the robot's course to be parallel to a wall typically happens after the robot is at more or less the correct distance, the alignment rules have higher priorities than the offset rules. We might consider eliminating rule 2 from this list since normally when the robot is close to a wall the W sensor is on anyhow. However, when the robot reaches a convex corner, this is not necessarily true and rule 2 is actually helps the robot round the edge. We might also consider combining rules 3 and 4 by using a disjunction (or). However, keeping them separate improves the modularity of the system. For instance, we might decide later that we will use a completely different behavior to center the robot in narrow corridors. In this case we might want to eliminate rule 3 while leaving the protective veering reflex (rule 4) unchanged.

COORDINATING MULTIPLE BEHAVIORS

Here we describe the design of part of a sophisticated mobile robot named Herbert. This robot is 18 inches in diameter and stands about 4 feet tall (see Figure 5). Herbert's overall goal is to collect empty soda cans. It starts by wandering around its environment and searching for cans with a structured-light ranging system. When it finds a promising candidate, it carefully approaches and aligns itself with the target. The robot then releases its arm and gropes around using local sensors to retrieve the can from wherever it is sitting. Once the robot has picked up the can, it slowly navigates back to its home position and deposits its trophy. Finally, the cycle repeats and the robot ventures forth once again on its quest. We concentrate here on just the manipulation phase and skip over the other subtasks.

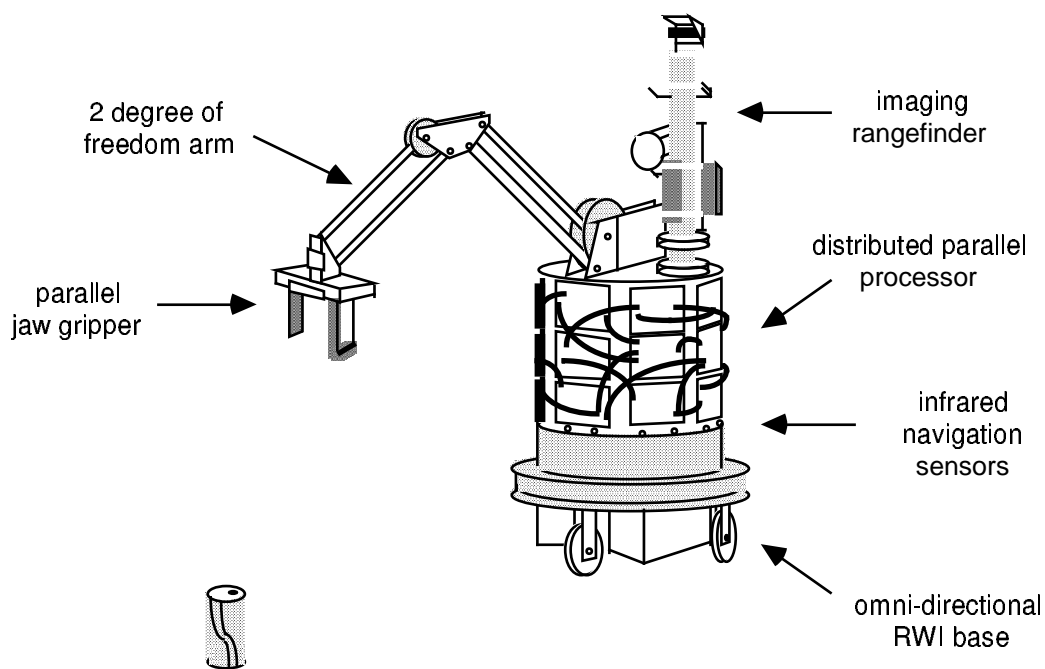


Figure 5 - Herbert's primary goal is to collect empty soda cans. It locates them with a laser light-striper, grabs them with a long arm, then uses proximity sensors and a compass to bring them back.

The most essential goal for our robot is to actually grasp a can. Crucial to the development of this subsystem is the arrangement of sensors on the hand (see Figure 6). These are positioned so they directly detect relevant situations. The types of situations and appropriate responses are described below. Our approach to this problem is to imagine a progression of increasingly complex situations, and then devise ways of reducing each case to the next easier one.

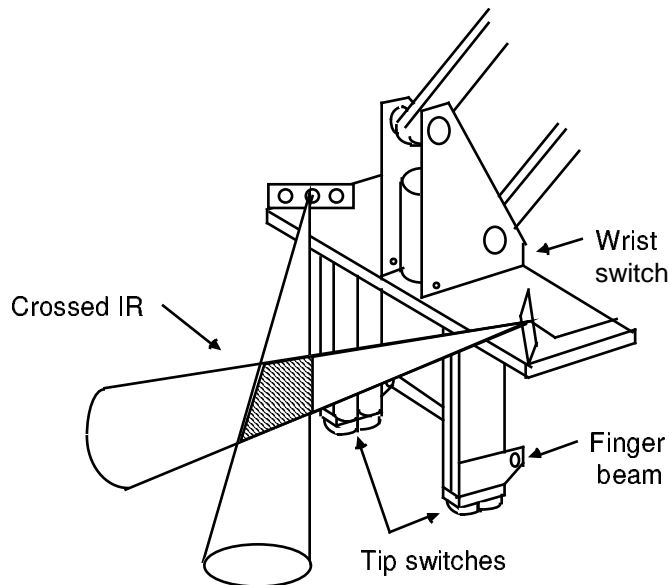


Figure 6 - The hand has a variety of local sensors which serve as the basis for our “matched filters”.

Let us start with the simplest case in which the can is already between the robot's fingers. This can be easily detected by looking for an interruption in a light beam near the back edge of the fingers. In this situation, the robot should obviously close its hand around the can. This is the first behavior in our control system: if anything is ever placed in the robot's hand, it grabs the object. Note that the physical construction of the robot's hand and the placement of the beam sensor already form a “matched filter” for graspable objects. Only things that fit between the spread fingers (a 5" gap) can be grabbed. Also, if the object is inserted from the front, it will be well positioned for gripping by the time the beam at the back is activated.

Next, let us consider the case in which a can placed directly in front of the hand. Here, a reasonable default strategy is to extend the arm forward until the grab reflex is actuated. This then becomes our second behavior. However, depending on the relative heights of the hand and the can, the robot might very well knock the can over. While many cans would fit in the hand even in this new orientation, they are much more prone to rolling across the table. To avoid this, we augment the robot's repertoire by adding a third behavior which monitors the crossed infrared proximity detectors affixed to the front of the hand. Again, the geometry is arranged such that this sensor directly detects objects that would collide with the upper part of the hand. Thus, when either of the two sensitive regions is occupied, the hand instead rises straight upward until the signal disappears. At this point, it has reduced the situation to one in which simply extending forward will successfully trigger the ordinary grasp reflex.

Normally, however, the can is some distance away from the hand at the start. Extending the hand straight forward while it is several feet up in the air is unlikely to achieve satisfactory results. Instead, it makes more sense to search for a surface which might be supporting a can. This can be done by starting the arm at the top of its workspace and then slowly moving straight down. Eventually, the robot's hand should touch either a table or the floor. We can detect this by watching for the small switches at the tips of the fingers to be activated. Since it is potentially damaging (and certainly unproductive) to push the fingers through the table top, when contact is detected we add a behavior which causes the robot to lift its hand slightly until it is no longer touching the surface. Now that a surface has been found, a good strategy is to cruise along parallel to it in search of cans. We do this by adding yet another behavior which drives the hand diagonally forward and down if the tip switches

have been active recently. The purpose of the diagonal motion is to keep the fingertips close to the surface. If there is some sort of pit on the table, we want the robot to be able to reach into it. Usually, however, the tip switches will quickly be activated again and the hand will rise up and move forward. Bouncing along the surface like this serves to reduce the situation to one of the earlier cases. Typically, the front infrareds will detect the can (if any) and cause the hand to rise over and grab it like before.

Note that this collection of behaviors essentially represents both cans and tables procedurally. Cans are things that fit between the fingers when the hand rises above them (as governed by the front beam sensors) and then extends forward. A number of other objects besides soda cans could also allow this sequence to run to completion. Examples are beer cans, baseballs, and small boxes. In this sense, we have really built a "graspable object" recognizer. Similarly, the robot's table representation is really based on functional properties. A table is a more or less flat horizontal surface which the hand can skim across. The robot does not really distinguish tables from floors but, since it starts at the top of the arm's workspace, the hand is more likely to encounter a table top first. The knowledge that tables are high up and that cans are often found on tables (or at least on supporting surfaces) is built directly into the rule set.

Suppose now, however, that all these behaviors are insufficient to keep the hand out of trouble and it manages to get stuck somewhere. One way to detect this situation is by noticing that the arm's angular configuration has not changed in a while. When this happens, we kick in a new behavior which attempts to retract the arm until it either reaches a special "home" position or a specific time limit is exceeded. This causes the robot to give up on grabbing the can, or to repeatedly tug on whatever is obstructing its arm. Thus, we ensure that the robot continues to make progress by monitoring a sensory-derived predicate which is indicative of that progress. Interestingly, we can use this same escape procedure for two other ways in which a groping trial may terminate. One situation is the successful grasping of a can. Here, we purposely freeze the arm until the fingers have had a chance to close. However, to the monitoring behavior this looks like a lack of groping progress and hence automatically initiates retraction. The other case is when the arm has travelled all the way to the edge of its physical workspace without finding a can. In this case, since it can no longer proceed in the commanded direction, it stops. Once again, this lack of progress triggers withdrawal of the arm.

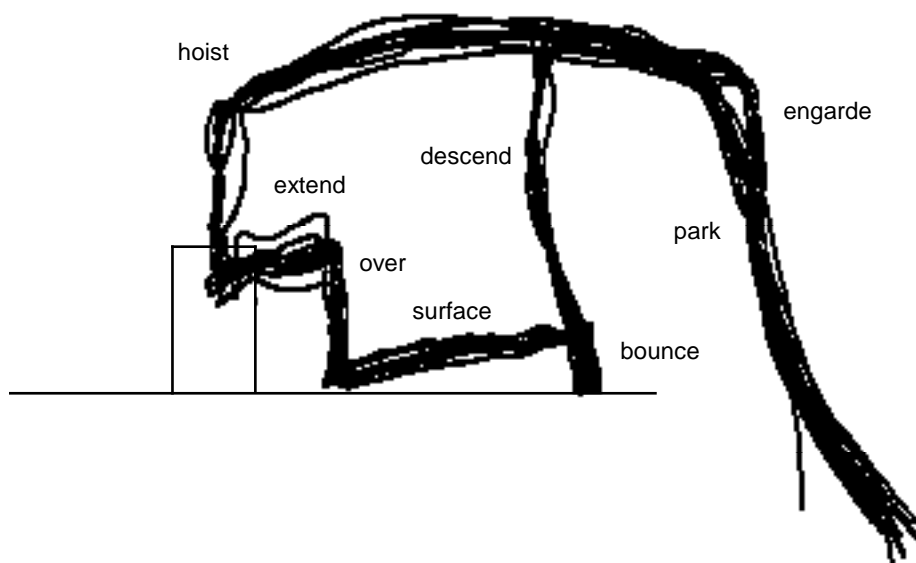


Figure 7 - The nominal environment consists of a can on a table top (thin lines). Here we show the path (thick lines) followed by the tips of the fingers during 10 consecutive runs on the real robot.

Herbert is very good at the grabbing task. Figure 7 shows some typical trajectories generated by the complete set of behaviors. At the start the ENGARDE and DESCEND behaviors locate the table top. Next, BOUNCE takes over and raises the hand and then lets SURFACE skim the hand along the table. However, before the next contact occurs the front IRs are activated which causes OVER to raise the hand to the top of the can and EXTEND to bring the can between the fingers. The GRAB behavior then comes on and closes the robot's hand around the can. Finally, HOIST and PARK bring the can back toward the robot's body. Note that events do not necessarily have to proceed in this manner. If, after the first bounce, contact is not re-established with the table, DESCEND will again become active in an attempt to better follow the surface. Similarly, if a sheer wall is encountered, OVER will raise the hand to near the top of the wall and then, with a little help from a behavior which monitors the force on the wrist, surmount it and allow the hand to continue groping at this higher level. Because the robot does not have a preset plan, there are never any overt "exceptions" or "contingencies". In this system, the trajectory is effectively programmed by the environment itself.

ADDING SUPERVISORY CONTROL

Consider the differences between riding a horse and driving an automobile. A horse will not run into a telephone pole at high speed. If you fall asleep in the saddle, a horse will continue to follow the path it is on. If you have two horses but only one rider, you simply have one horse follow the other. In general, horses are much smarter than automobiles. This arrangement provides a good example of partnership between man and "machine": the horse takes care of the details of motion while the human provides overall guidance. In the previous sections we described the design of competent animal-like robots. However, for some applications these may be too far in the opposite direction from cars; more like wild horses that do what they feel like and are oblivious to outside commands. In this section we show how to tame the creature by adding supervisory control to a reactive multi-agent system and describe an actual robot ("Mister Ed") built along these lines.

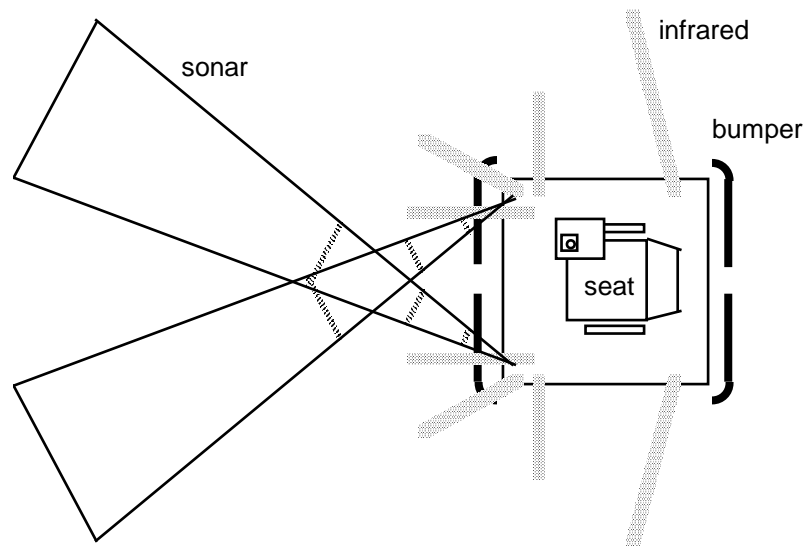


Figure 8. As shown in this overhead view, our mobile robot has 3 sensory modalities. The infrared proximity sensors are used for local navigation and obstacle avoidance, the sonars are used for following people, and the bumpers are used for collision detection.

As before, the control system of this robot consists of a collection of situation-action rules. The input to this rule system comes from a series of sensors mounted around the periphery of the robot. As shown in Figure 8, the robot has three different sensory modalities. At the front and rear of the robot there are bumper switches (black bands) to detect any collisions that may occur. Each bumper is divided in half to allow lateralization of the contact. For local navigation there are 8 very long range proximity detectors arrayed around the robot (gray lines). These units ("IRs") emit infrared energy in a tight beam and look for any reflection above a certain intensity threshold. The sensors have been angled downwards to insure that any object sensed is actually between the robot and the floor intersection point. Finally, there are two Polaroid sonar sensors mounted at the front of the robot such that their radiation patterns overlap. The sonar echoes are classified into one of 4 distinct qualitative regions (dotted lines show divisions) to assist the robot in following objects.

A summary of the various behavior modules used by our system is given below. The driver's directions are incorporated into the control system by disguising them as the outputs of two normal rules (**Joy-Trans** and **Joy-Turn**). Thus, the driver can override some of the robot's innate tendencies yet there are other behaviors which can override him. As before, the complete set of rules are listed in order of ascending priority. For instance, the output (if any) of the **Retreat** rule completely overrides any output produced by the **Trek** rule. Notice that the behaviors which control the robot's translation are separated from those that control its rotation. This is because sometimes the relative priorities of related behaviors are different. For instance, if the robot hits an obstacle, **Bounce** insists on backing up no matter what the driver commands via **Joy-Trans**. Yet, since **Joy-Turn** takes precedent over **Turn**, the robot is willing to let the operator choose a particular direction of travel by deflecting the joystick.

Translation behaviors (in order):

- Approach** - Go forward if an object is detected beyond the optimal sonar range.
- Trek** - Go forward if any of the side IRs have been active recently.
- Retreat** - Go backward if an object is detected in the near sonar range.
- Joy-Trans** - Obey all of the driver's forward and backward commands.
- Bounce** - Go forward or backward a while depending on which bumper was hit.
- Halt** - Prevent travel in the direction corresponding to the impacted bumper.
- Fend** - Prevent forward travel if any of the front IRs are active.

Rotation behaviors (in order):

- Align** - Turn toward the the nearest object detected by the sonar.
- Parallel** - Suggest turning to make both side IR sense the wall if one did.
- Turn** - Turn away from side of the recently hit bumper.
- Stymie** - Turn left if all the front IRs are active.
- Thread** - Turn to unblock a single front IRs.
- Veer** - Turn to unblock a single diagonal corner IRs.
- Twist** - Turn toward the side on which a persistent IR signal recently disappeared.
- Joy-Turn** - Obey the driver's left and right turn commands.
- Point** - Prevent any turn if a bumper is hit.

The outputs from these collections of production rules control the motors on a two-wheeled base. In this robot, we have divided the motion control problem into a forward/backward velocity component and a left/right velocity component. These are in essence two different "resources", each with its own set of behaviors as shown above. In other robots, we have further subdivided control by adding "speed" resources and special velocity regulating rules for each type of motion. Here, we just quantize each degree of freedom to either positive, negative, or no motion (i.e. 2 bits). Several times each second, Mr. Ed runs through all the active rules and updates his four motion control bits accordingly.

It usually not desirable to run all the robot's rules at the same time. For instance, the goals of person tracking and wall following (e.g. **Align** versus **Parallel**) can interfere with each other. Such interactions become an especially serious concern as the size of the rule base grows. While some of this disorder is resolved by breaking down the control of the robot based on different actuator resources, we address the fundamental problem by segregating rules into small groups. Each "competency module" constitutes a toolbox for achieving a particular goal. Since they are not intended to be used together in arbitrarily large groups, as a first approximation inter-rule dependencies need only be considered within each module.

This suggests another way in which the human driver can influence the robot, other than being just another rule in the system. We provide a series of toggle switches which allow the operator to selectively enable and disable various groups of rules. This is how the person tells the robot what its current "goal" should be. However, each user-level goal is usually comprised of a number of sub-goals. Thus, there is generally not a one-to-one mapping between switches and behavior groups. Figure 9 shows this breakdown. Also, certain basic behaviors, such as the joystick interpretation rules, are always active (dark bordered box) and others are active anytime the robot is allowed any control at all (gray bordered box). In our robot, these groups correspond roughly to the universal goals of obedience and self-preservation.

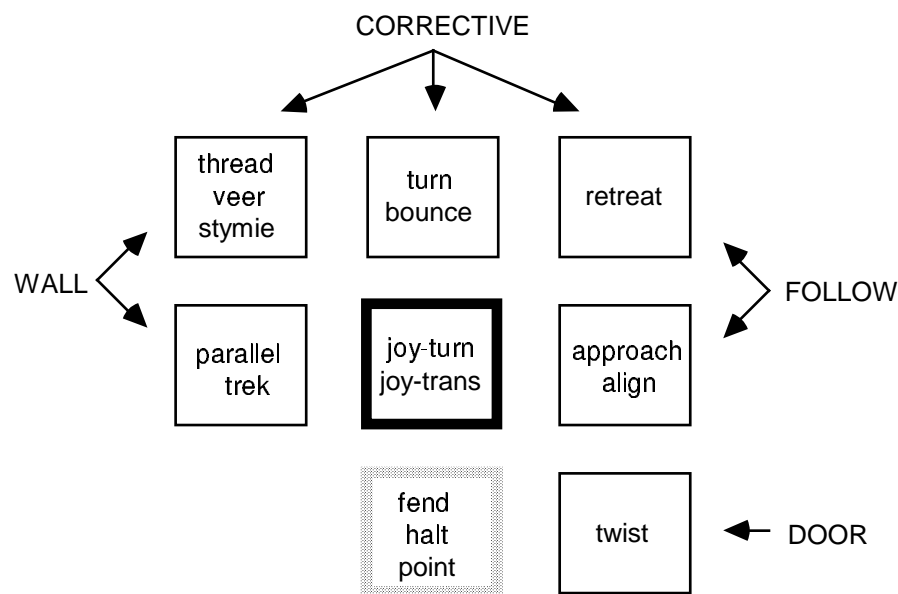


Figure 9. The robot's situation-action rules are broken up into a number of functional groups. The robot can be authorized to perform a specific activity by enabling certain of these groups. Some groups (heavy bordered boxes) are always active, no matter what activity is selected.

The use of this partitioned system is best shown through several examples. Suppose, at the start, that all groups except the joystick repeaters and basic protective behaviors are disabled. The operator is basically free to drive the robot in any direction desired. However, if the front IRs detect an obstacle, the robot will refuse to continue going forward. Like a horse, it balks at this command. In certain cases, such as traversing a narrow doorway, the operator may decide that the robot is being too cautious and then disable the protective behaviors to obtain complete control. On the other hand, he may instead wish to delegate more authority to the robot by enabling the corrective steering behavioral groups. The rider now retains control of most of the forward/backward motion of the robot, but does not have to worry about veering around obstructions. He simply tells the robot to go forward and it takes care of the moment-to-moment heading changes required.

The rider may eventually get tired of driving altogether and give the robot even more control by activating the person following routines. This is useful in large open spaces where the final destination is unknown. Alternatively, the rider may instead decide to travel along some particular natural path in the environment, such as a the edge of a room or corridor. To start the robot in some particular direction, the operator enables the appropriate rule groups and then drives the robot until it is more or less aligned with the wall or corridor it is supposed to follow. At this point the driver can relax and the robot will continue along this environmentally defined path indefinitely. He is free to fall asleep in the saddle if he wishes. In this case, the robot is acting as a smart peripheral. A similar approach could have been used on the can collecting robot to emulate the convenience of the "grab-it" button found in many video-games. By relying on a collection of local procedures, the high level controller need not worry about actually moving the agent's hand or about a achieving a firm grasp on the desired object.

When running in the autonomous wall following mode, the driver will occasionally resume control for a moment. The usual reason is to a make strategic navigation choice such as which corridor to follow at a branching, or whether to go through an open door. Yet, even some of this can be delegated to the robot. If the door finder routine is enabled the robot will execute a 90 degree turn at the next aperture it sees, and then continue forward using its local navigation routines. Thus, the driver never has to take real-time control of the robot, he merely modifies its reflexes so the appropriate action will automatically be taken in each situation which is likely to occur. This is analogous to advice such as "the play is to second" muttered during a baseball game. All the computation intensive deliberation can be performed off-line and then compiled down for later use should the contingency arise.

SUMMARY

We have described how a number of partial representations embedded in parallel control paths can be used to control a mobile robot. This approach allows the incremental development of a system by the accretion of successive performance layers and is operationally robust because of its flexible activity scheduling. We then turned to practical matters and showed how to create basic behaviors by defining a number of easily extractable partial representations and then devising appropriate control actions for each key situation. Next, we discussed how each behavior contains some grain of expertise concerning a particular subtask and then explained how to make these individual behaviors cooperate to achieve the overall task. Finally, we examined several ways in which outside guidance could be provided to such a distributed system.

In the course of explaining the design principles we have also covered a variety of different types of behaviors. Looking back on these, we find that they basically fall in to four classes. First, there is the collection of primitive "Avoidance" behaviors. These take care of collision with obstacles, escaping from predators, and maintaining the appropriate the separation from the creature's peers. On top of these, there are usually a number of "Exploratory" behaviors. These let the robot do things like map its environment or forage for "food". Beyond these, there is typically a class of "Seeking" behaviors which cause the robot to go to some predefined location or to approach some type of target when it is visible. Finally, although not discussed here, there is also usually a collection of "Social" behaviors. These govern aspects such as "courtship", ritualized combat, cooperative building (e.g. of nests), and team hunting (i.e. in packs). You might want to keep these four broad classes in mind when designing behaviors for your own robot.

While you can spend years developing a theoretically perfect set of control algorithms, chances are they will not work in the real world. Our approach has been along engineering lines instead: we first find out what works then go back and try to extract useful generalizations from it. We believe empirical studies such as this to be of great value in mobile robotics.