Asynchronous Distributed Control System For A Mobile Robot

Rodney A. Brooks, Jonathan H. Connell

Asynchronous distributed control system for a mobile robot

Rodney A. Brooks and Jonathan H. Connell

MIT Artificial Intelligence Lab
545 Technology Square
Cambridge, Mass, 02139

Abstract

In this paper we describe the *subsumption architecture*, a design methodology for multiprocessor control systems. The architecture is based on a new decomposition of problems into task-achieving behaviors. This decomposition allows us to incrementally construct more competent robots by adding new behaviors. We demonstrate this by showing how two different control systems can be built on top of the same set of core behaviors. We have implemented both systems and have run them on a real mobile robot. The details and performance of each will be discussed.

The architecture

In our research we have adopted a particular strategy for dealing with the concurrent processes necessary to run a mobile robot. This strategy is to combine a number of simple *independent* task-achieving behaviors to yield a more complex behavior. As pointed out in [1], most control systems decompose a problem into a sequential set of subproblems linking the input to the output (Figure 1a). The approach we will consider here decomposes the problem into a parallel set of control systems, each of which goes all the way from input to output (Figure 1b).

We chose this approach for a number of reasons. First, it allows us to build up a complete control system incrementally. Typically we start with a small number of general purpose behaviors that work for most of the situations encountered. Later we patch up the system by adding special purpose behaviors to handle the cases in which the generic strategies fail. In this way we end up on the winning side of the 80–20 phenomenon (the observation that 80% of the results can be obtained with only 20% of the work). Furthermore, a multiple agent system allows us to debug each individual behavior in isolation from the others. This is especially important as systems become more complex. Finally, this approach yields a robust system whose performance degrades

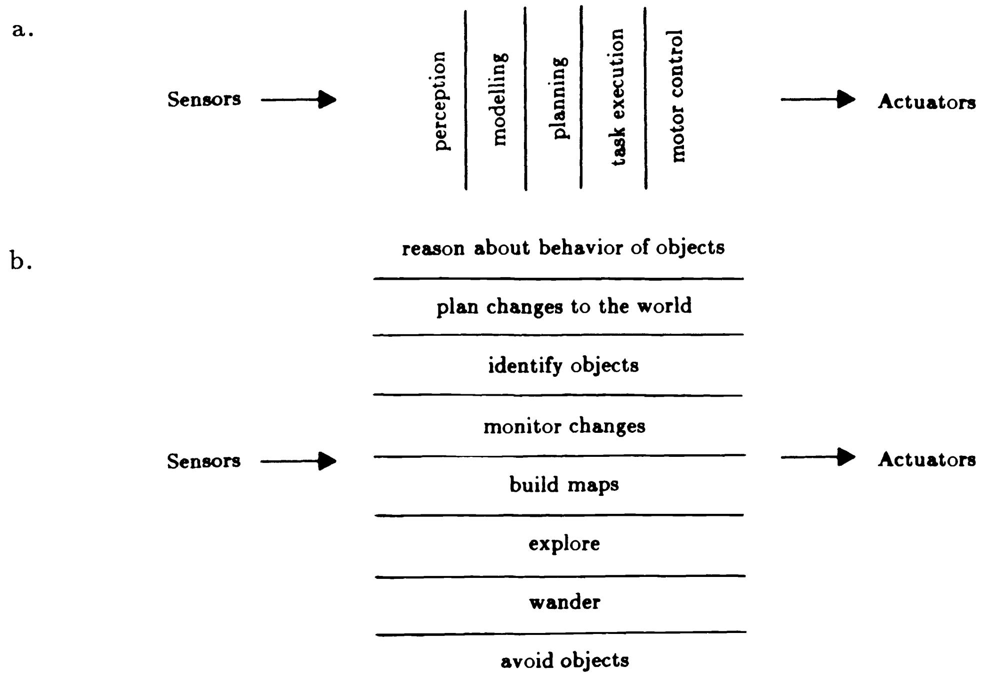

Figure 1. Slicing a control system. a. Traditional decomposition. b. Breakdown in terms of task-achieving behaviors.

a.

```
(defmodule collide
  :inputs (map)
  :outputs (halt)
  :states
    ((nil (event-dispatch map monitor))
     (monitor (conditional-dispatch (dangerous-motion-p map 5.0)
                                    quit
                                    nil))
     (quit (output halt hi)
           nil)))
```

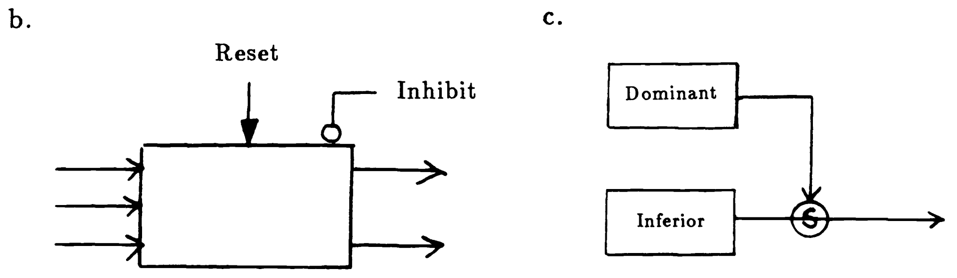

Figure 2. Brooks's modules. a. Each module is a finite state controller with a small number of geometric functions. b. The reset input puts the module in state *nil* while the inhibit input prevents outputs from being transmitted. c. The message coming in to a suppressor node takes the place of the message passing through.

gracefully. That is, even though certain behaviors may fail to achieve their tasks, the performance of other behaviors is not impaired.

Modifications made in the course of learning also benefit from these features. For instance, to navigate in an environment the robot can start by just knowing particular paths from one place to another. Because the system is incremental it doesn't have to know the complete layout of the space from the start. Furthermore, each path can be debugged (by utilizing different landmarks) or improved (by finding shortcuts) independent of the rest of the paths. Finally, the system is robust enough to allow the robot to navigate fairly well even if certain paths become blocked. We describe a system like this in [2].

We make these multi-agent control systems out of smaller units called *modules*. Each module has a certain number of inputs and outputs for communicating with other modules. Inside a module has a finite state controller and a few simple geometric functions (Figure 2a). In addition there are two distinquished inputs to the module: reset, and inhibit (Figure 2b). When an inhibit signal is present the module is prevented from issuing any outputs. This provides a handy method for temporarily disabling a particular behavior. The other input, reset, forces the internal finite state machine back to its initial state. This can be used for temporarily synchronizing two modules.

There is a third special form of interaction, *subsumption*, from which the architecture gets its name. The idea is that one module can commandeer another module's input wire and substitute its own message in place of whatever was being transmitted over the wire. This is represented as shown in Figure 2c. Subsumption lets us patch up a control system by allowing smarter behaviors to take over from default behaviors when appropriate.

The lowest level

The lowest level of control shows how several agents can cooperate to achieve a task. In the MIT mobile robot the most primitive behavior is "don't hit things". Figure 3 shows the control system for doing this. The robot uses a ring of 12 ultrasonic range-finders to sense obstacles around it. As suggested in [3], we model each obstacle by a repulsive force which falls off as the square of the distance of the obstacle from the robot. The FEELFORCE module sums the forces sensed by each sonar and generates a composite force vector. The RUNAWAY module monitors the magnitude of this force and determines if anything has gotten too close. If so, it generates a command to run away in the direction of the resultant force vector. The movement command, a heading and a distance, are sent to two modules which control the robot's motors. The TURN module first rotates the robot in place until it is oriented properly and then passes control to the FORWARD module which makes the robot travel the specified distance.

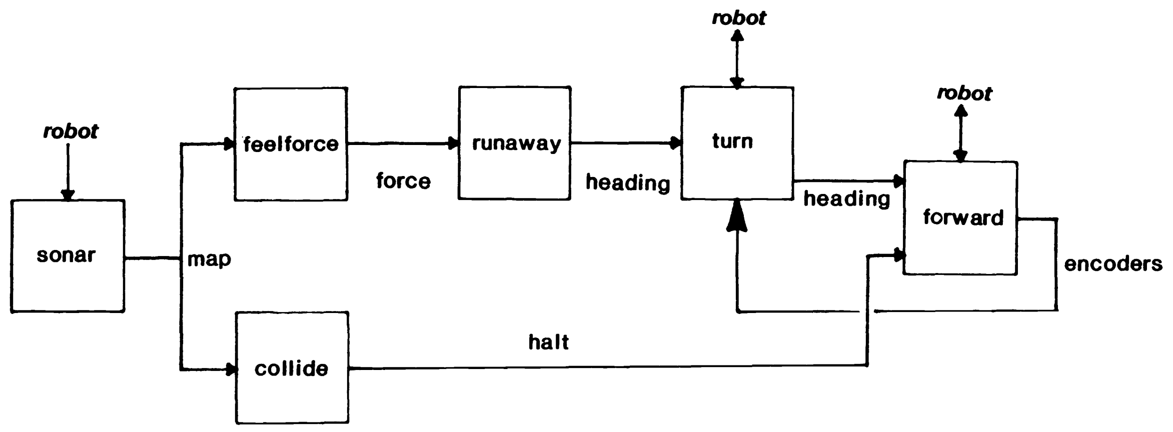

Figure 3. The zeroth level of control tries to keep the robot from hitting things.

Running in parallel with the "flee" behavior is a "freeze" behavior. This is mediated by the COLLIDE module which continuosly monitors just the front two sonars to see if there is something in the robot's path. If so, COLLIDE stops any forward motion currently in progress by sending a "halt" message to the FORWARD module. While this signal is present the robot can not move forward but it is allowed to turn in place. Thus, when someone jumps into its path the robot stops suddenly, and then takes evasive action by first turning away and then going forward.

A wandering robot

The zeroth level described above protects the robot but does not make it do anything particularly interesting. For this reason, after we had thoroughly debugged the system described above, we decided to give the robot an exploratory behavior. The interesting part is that we left the original control system intact and built the new control system around it. As shown in Figure 4, the new layer was allowed to spy on the connections between modules in the old layer but none of these connections were changed. For instance, to drive the robot in a particular direction, the wire from RUNAWAY is not cut, rather the AVOID module suppress this output. This method of combination has the feature that if the new layer ever stops sending commands, control will automatically revert to the lower level RUNAWAY behavior. During all this, of course, the COLLIDE module is still running and functions exactly as before.

The first level layer of control, when combined with the zeroth, imbues the robot with the ability to wander around aimlessly without hitting obstacles. This control level relies in a large degree on the zeroth level's aversion to hitting things. In addition, it uses a simple heuristic to plan ahead a little in order to avoid possible collisions which otherwise would need to be handled by the zeroth level. Every 10 seconds or so, the WANDER module generates a new heading for the robot which AVOID then combines with the result of the force computation from the zeroth level. This produces a modified heading which usually points in roughly the right direction, but is perturbed to avoid any obvious obstacles. Thus, the computation performed by AVOID implicitly subsumes the computations of the RUNAWAY module.

The next level, level two, is meant to add a more directed exploratory mode of behavior to the robot by selecting interesting places to visit. In our case, interesting places are corridors which the FREESPACE module finds by picking the longest sonar return above some threshold (eventually we will use vision to find corridors). Additional modules then provide a means of position servoing the robot along the corridor despite the presence of local obstacles in its path. The wiring diagram for this level is shown in Figure 5. Note that it is simply Figure 4 with some more modules and wires added.

Alot of the structure of this level is concerned with calming the robot down so that it can examine its environment. This will be especially important when we convert to the vision-based corridor finder since the stereo routine we intend to use [4] takes 17 seconds to process an image. If the robot moves in this interval the data obtained will be useless, thus we only look for freeways when the robot is stationary. Detecting when the robot is idle is the primary function of the STATUS module. This module monitors the TURN and FORWARD modules and maintains a status output which sends either *hi* or *lo* messages to indicate whether the robot is busy. In addition, at the completion of every turn and drive combination, STATUS sends out a combined set of shaft encoder readings which indicate how far the robot has actually moved. The WHENLOOK module, however, only monitors the "busy" line and, whenever the robot has been sitting idle for a few seconds, inhibits WANDER to keep the robot from straying from its current

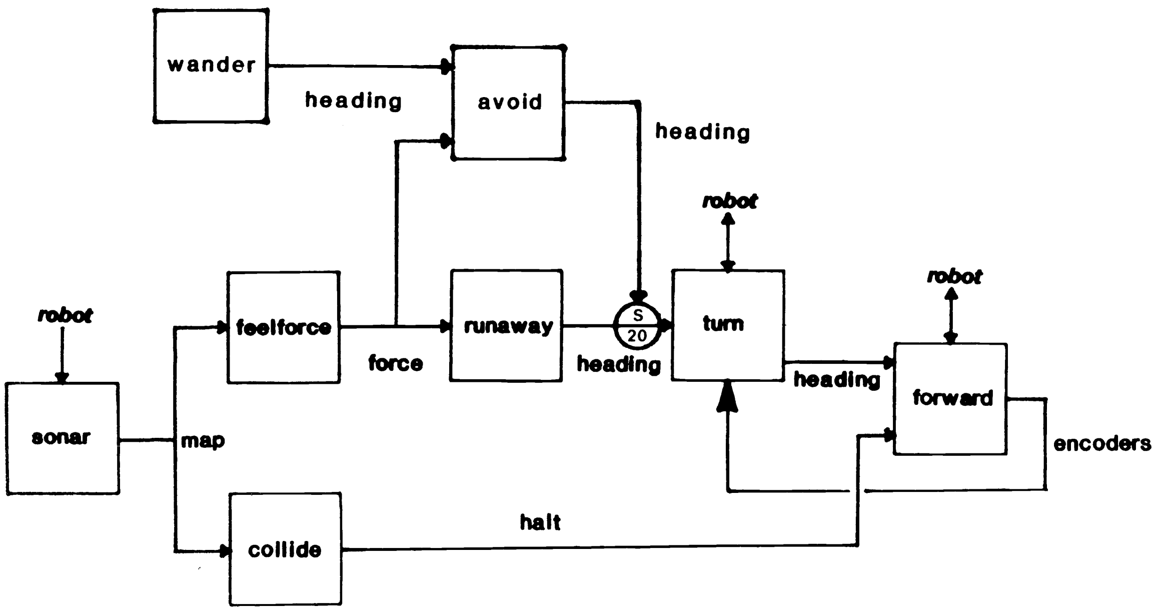

Figure 4. The first level control system chooses random headings every 10 seconds. Note that it is simply the lowest level control system with some more modules and wires added.

location. The LOOK module then initiates the vision processing, waits for a candidate freeway, and, if the candidate passes certain tests, passes it on to the PATHPLAN module.

The other modules in Figure 5 control the robot as it trundles down a freeway. INTEGRATE accumulates reports of motions from the STATUS module and sends its most recent result out on its "integral" line. INTEGRATE is initialized by LOOK to ensure that the robot will know how far it has moved since from where it took its stereo pictures. This is important since sometimes impulses from RUNAWAY will perturb the robot while these images are being processed. The last module in the chain, PATHPLAN, takes the goal specification provided by INTEGRATE (an angle to turn and a distance to travel) and attempts to reach that goal. To do this it suppresses the input to the AVOID module thus substituting its headings for the headings normally supplied by WANDER. Therefore, so long as the higher level planner remains active, the random wanderings of the robot will be suppressed. Finally, when the position of the robot is close to the desired position (the robot is unaware of control errors due to wheel slippage etc., so this is a dead reckoning decision) PATHPLAN terminates. Sometimes, however, the robot is unable to reach the goal. For this reason there is one more module, GIVE-UP, which stops PATHPLAN if it hasn't gotten to the goal in 2 minutes.

A wall-hugging robot

After experimenting with the system described above, we decided to implement a somewhat different control scheme. The first part of the new system causes the robot to track along walls at a fixed distance instead of wandering randomly. The second part of the system is much like the old one except that now the robot looks for doors instead of corridors. The important point is that we *retained* the same zeroth level as used by the other system. This clearly illustrates the modularity available with the subsumption architecture.

The method the second level uses to follow walls is particularly interesting because it is based on a very incomplete model of space. First, the WALL module generates a fixed strength attractive vector at an angle (usually 120 degrees) relative to the repulsive force. This is then added to the original repulsive vector from FEELFORCE by the AVOID module (Figure 7). When the robot is at the appropriate distance from the wall, the repulsive and attractive components normal to the wall cancel out. This leaves only the tangential component which causes the robot to move along the wall. If the robot gets too close, the repulsive force becomes stronger and the robot veers away. Similarly, if the robot gets too far from the wall, the attractive force wins and it angles inward. This is much different from taking the raw sonar readings, fitting straight lines to the freespace diagram, and then planning a trajectory toward the closest wall found. Using the potential field method, the robot can follow walls without having any internal model of what a wall is.

The top layer in this control system is shown in Figure 8. As before, this level implements a more directed exploratory behavior. While following walls allows us to accurately map the room the robot is in, unfortunately it also acts to confine the robot to that

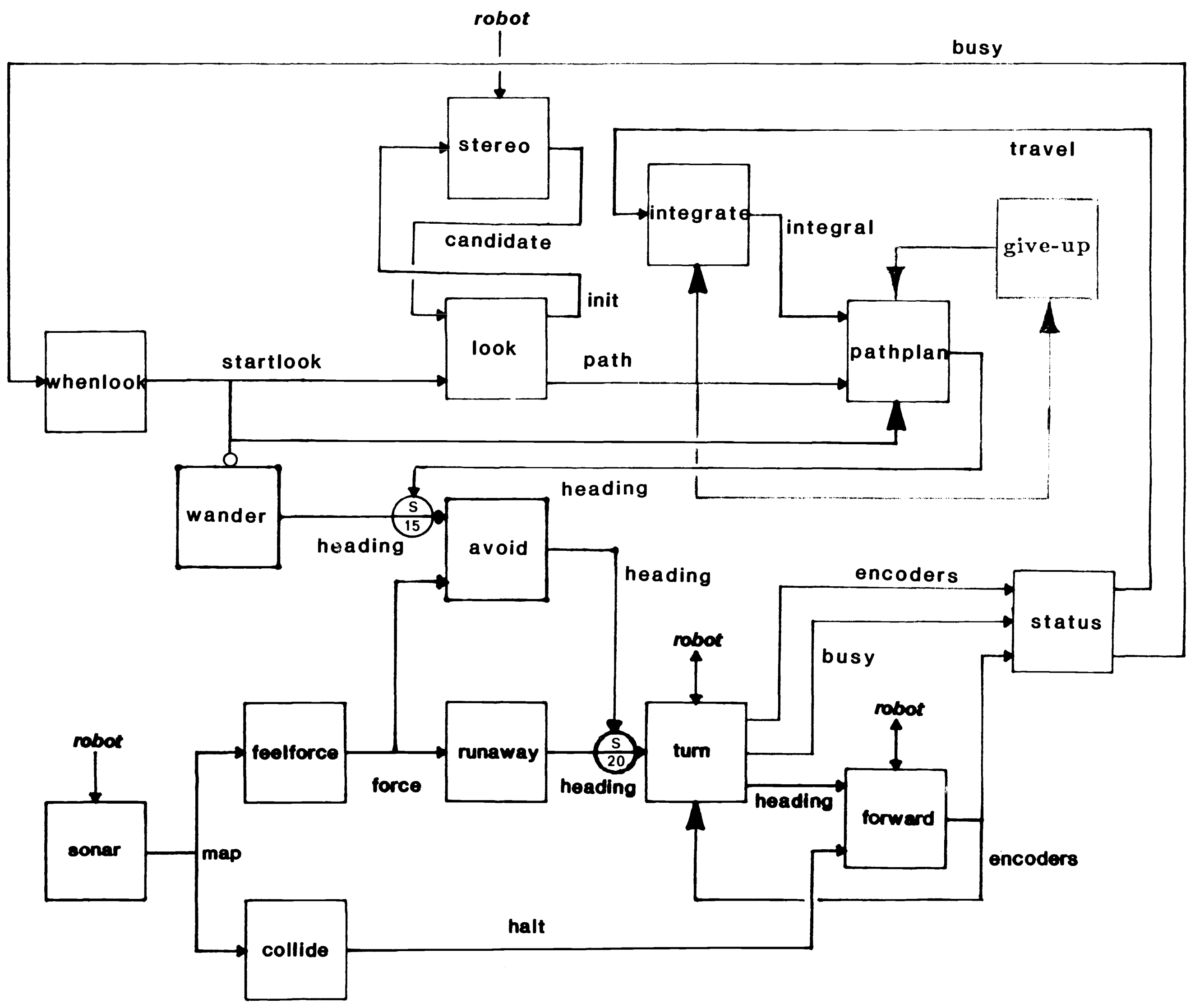

Figure 5. The second level looks for corridors of free space. The zeroth and first layers still play an active role during normal operation of this layer.

room. Due to the width of the sonar beam, even though a door causes a gap in the wall, this has very little impact on the repulsive force felt by the robot. In general, unless they are very wide, the robot ignores all doors when it is in the wall-following mode. The new level described below counteracts this by looking especially for doors and, when one is found, forcing the robot to go through it:

As with walls, doors are found using a very incomplete model. The DOOR module operates by looking for a long sonar return flanked by two close ones. This model is only accurate over a certain range of distances: if the robot is too far from the door it won't see any long returns (the sides of the door reflect the sonar beam), while if it is too close it will see several long returns through the door instead of only one. This new level therefore relies on the previous wall-following level to get the robot close enough to the walls that a door can be found, and to keep it far enough away that the door generates the qualitatively right kind of sonar pattern.

Note that this level not only uses the behavior of the lower level to help it recognize doors, it also uses some of the lower level

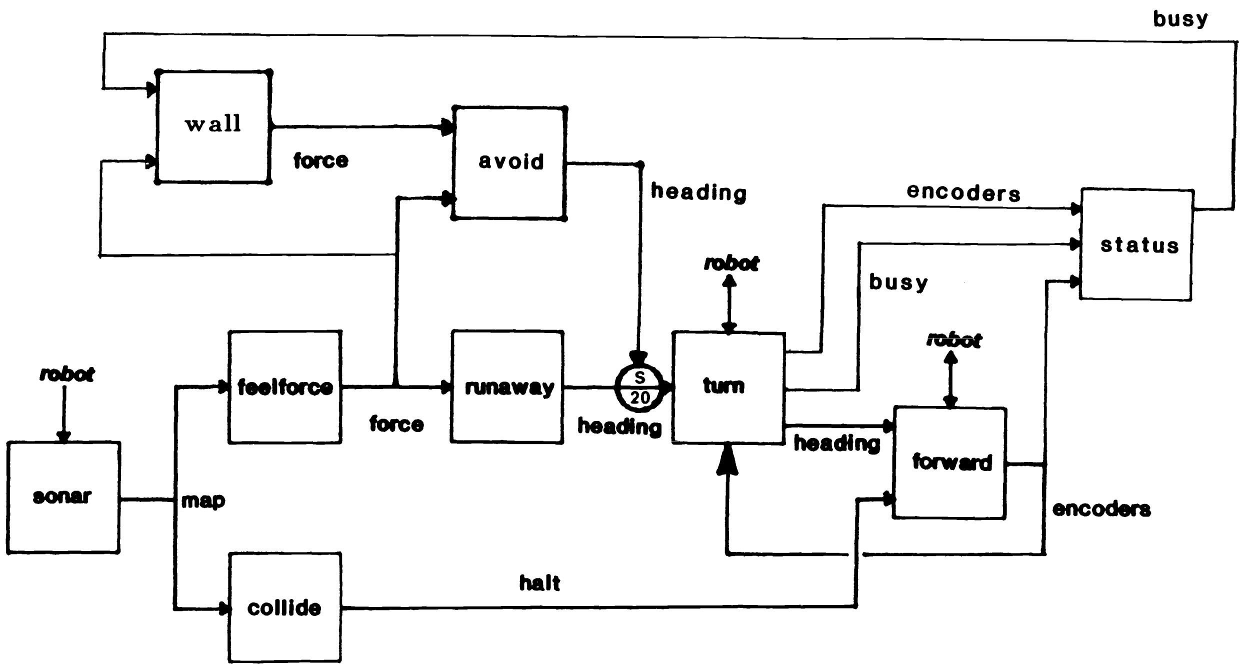

Figure 6. Level 1 in this system causes the robot to follow walls.

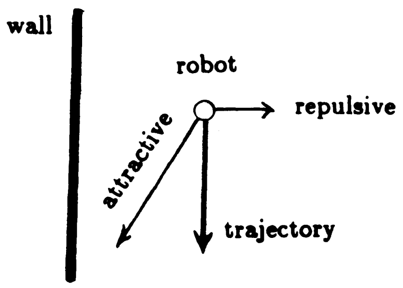

Figure 7. The robot has no internal model of a wall. Rather, it determines the direction to travel by adding the force vector generated by the obstacles to another vector at a fixed angle to the first.

modules to get it through the door. If the robot just servoed on the door finder vector, it would never get through because the target would vanish as the robot got closer. Therefore, once DOOR has found a candidate it tells INTEGRATE to remember the heading and distance to the door. As the robot moves, INTEGRATE constantly updates these estimates and sends them to SEEK which transforms them into attractive goal vectors for AVOID. As before, AVOID mixes the goal vector with the result of FEELFORCE to allow the robot to simultaneously pursue its goal and avoid collisions. This is crucial since SEEK's headings usually don't take the robot through the exact center of the door. Finally, as with the previous control system, there is a GIVE-UP module which stops the robot if it is unable to get through the door after trying for a while.

Discussion

Figure 9 shows how the robot behaves under each control system. While the results shown here are simulations, these control system have been tested on a real robot with similar results. As shown in Figure 9a, the robot starts out at the entrance to a suite of four rooms (unfurnished as yet). Figure 9b shows the wander system in action. In this example the robot starts out following random headings until it gets to the edge of the large room. It was unable to tell that it was on a freeway in this area because the sonars we use to find freeways have a 30 degree resolution. Thus, even when looking straight along the corridor, the robot always

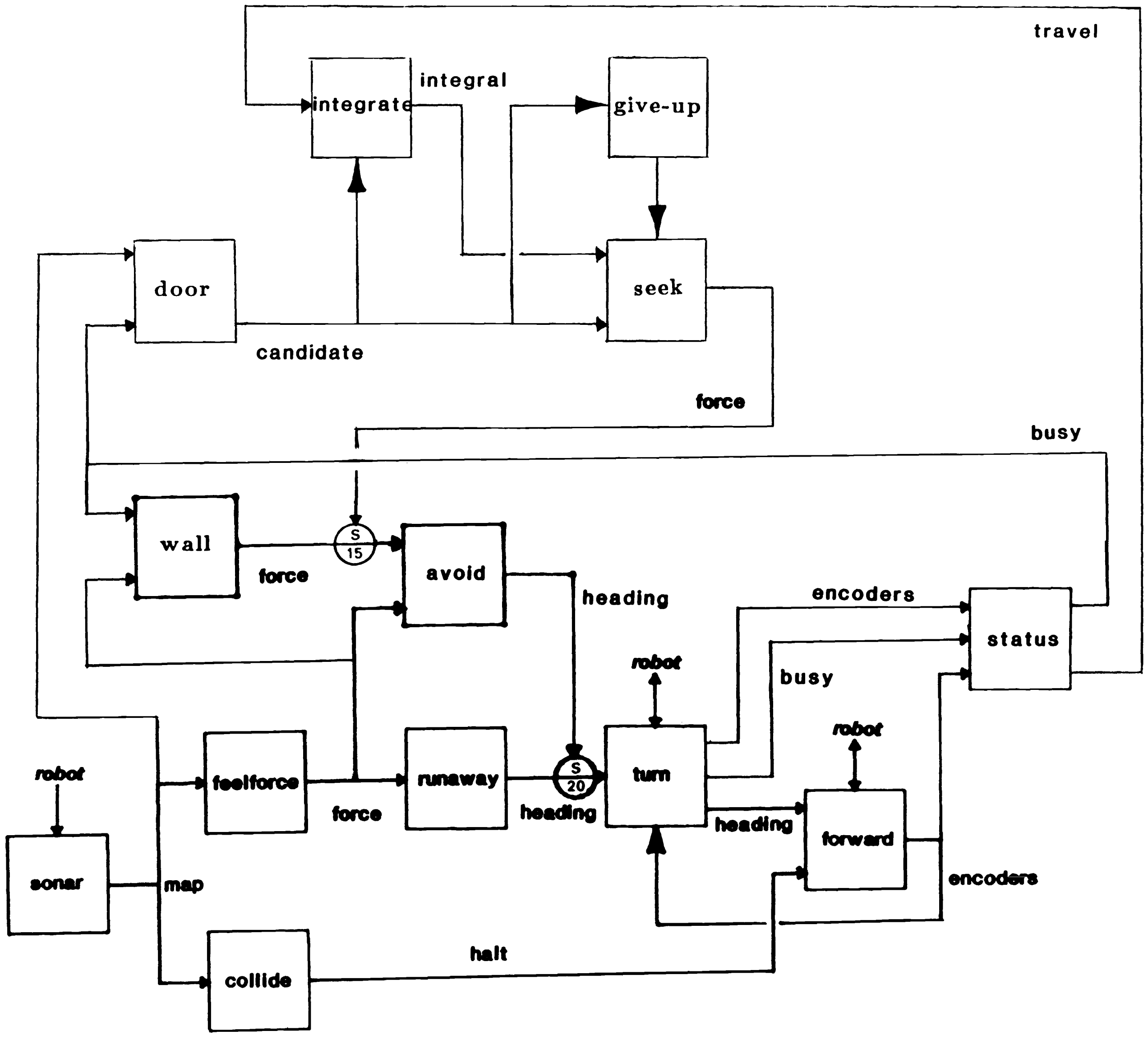

Figure 8. The second level control system helps the robot explore its surroundings by finding doors.

got returns from the two side walls. However, when it fortuitously reaches the end of corridor, it does detect a long clear path and heads off diagonally across the room. When the robot gets to the far side it notices that there is a long area of freespace behind it (the one it just traversed). Since it has no memory of where it has been, it decides to head back in the same direction it came. This oscillation will continue indefinitely until some external force disturbs the robot or until steering errors accumulate and cause it to go in a different direction.

The behavior of the wall following system is qualitatively different as shown in Figure 9c. Here, the robot starts out by following the right wall of the corridor. As with the other system, it only gets long sonar readings when it gets near the end of the corridor. At a certain point there is a long return flanked by two shorter ones which causes the robot to pass through this "door". It ends up near the center of the large room and then spirals out toward the right wall again. As can be seen, it continues to track along walls and go through doors and manages to explore a good part of its space. Like the other system, however, it has no memory and thus sometimes oscillates, in this case back and forth through doors (e.g. the upper right corner). Also, while it can find many doors, some, such as the one between the top two small rooms, elude it. Once again this is due to the coarse angular resolution of

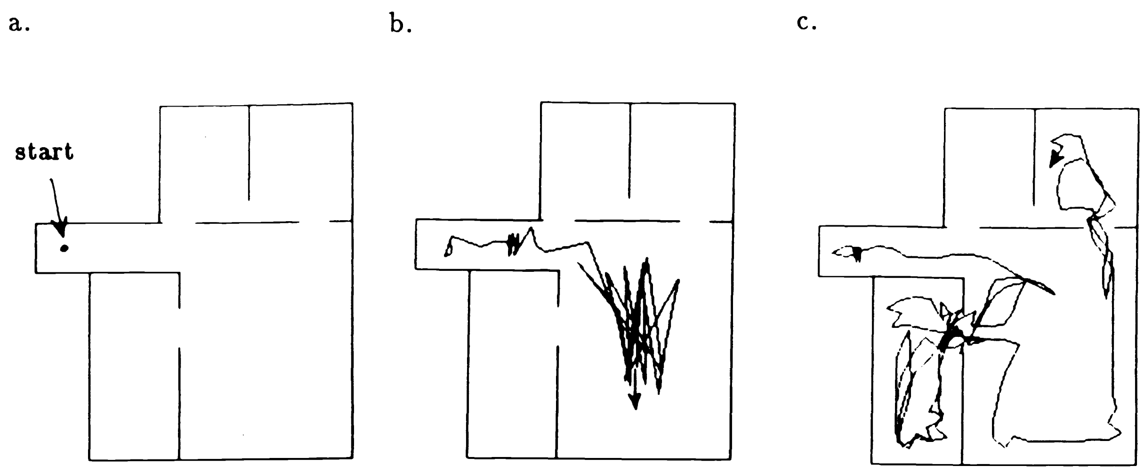

Figure 9. The path followed by the robot. a. The starting point in a suite of four rooms. b. The wandering system. c. The wall following system.

the sonar.

In this paper we have described two mobile robot control systems built using the subsumption architecture. These systems work both in simulation and on a real robot. Furthermore, while the systems exhibit different behaviors they both "evolved" from the same initial control system. The subsumption architecture models this evolution by the accretion of succesive control layers.

Acknowledgements

Support for this work was provided in part by an IBM Faculty Development Award, in part by a grant from the Systems Development Foundation, and in part by the Advanced Research Projects Agency under Office of Naval Research contracts N00014-80-C-0505 and N00014-82-K-0334.

References

1. Brooks, R. A., "A Robust Layered Control System for a Mobile Robot", *IEEE Journal of Robotics and Automation*, RA-2(1), April 1986.
2. Brooks, R. A. and Connell, J. H., "Navigation Without Representation", (in preparation).
3. Khatib, O., "Real-Time Avoidance for Manipulators and Mobile Robots", *International Journal of Robotics Research*, 1986.
4. Grimson, E. L., "Computational Experiments with a Feature Based Stereo Algorithm", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-7, January 1985.