A Robust Layered Control System For A Mobile Robot

RODNEY A. BROOKS, MEMBER, IEEE

Abstract—**A new architecture for controlling mobile robots is described. Layers of control system are built to let the robot operate at increasing levels of competence. Layers are made up of asynchronous modules that communicate over low-bandwidth channels. Each module is an instance of a fairly simple computational machine. Higher-level layers can subsume the roles of lower levels by suppressing their outputs. However, lower levels continue to function as higher levels are added. The result is a robust and flexible robot control system. The system has been used to control a mobile robot wandering around unconstrained laboratory areas and computer machine rooms. Eventually it is intended to control a robot that wanders the office areas of our laboratory, building maps of its surroundings using an onboard arm to perform simple tasks.**

I. Introduction

A CONTROL SYSTEM for a completely autonomous mobile robot must perform many complex information processing tasks in real time. It operates in an environment where the boundary conditions (viewing the instantaneous control problem in a classical control theory formulation) are changing rapidly. In fact the determination of those boundary conditions is done over very noisy channels since there is no straightforward mapping between sensors (e.g. TV cameras) and the form required of the boundary conditions.

The usual approach to building control systems for such robots is to decompose the problem into a series (roughly) of functional units as illustrated by a series of vertical slices in Fig. 1. After analyzing the computational requirements for a mobile robot we have decided to use *task-achieving behaviors* as our primary decomposition of the problem. This is illustrated by a series of horizontal slices in Fig. 2. As with a functional decomposition, we implement each slice explicitly then tie them all together to form a robot control system. Our new decomposition leads to a radically different architecture for mobile robot control systems, with radically different implementation strategies plausible at the hardware level, and with a large number of advantages concerning robustness, buildability and testability.

Manuscript revised February 3, 1986. This work was supported in part by an IBM Faculty Development Award, in part by a grant from the Systems Development Foundation, in part by an equipment grant from Motorola, and in part by the Advanced Research Projects Agency under Office of Naval Research contracts N00014-80-C-0505 and N00014-82-K-0334.

The author is with the Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 545 Technology Square, Cambridge, MA 02139, USA.

IEEE Log Number 8608069.

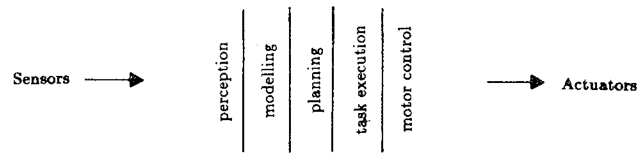


Fig. 1. Traditional decomposition of a mobile robot control system into functional modules.

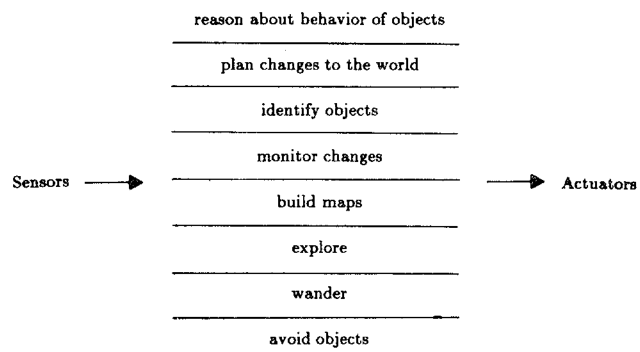


Fig. 2. Decomposition of a mobile robot control system based on task-achieving behaviors.

A. Requirements

We can identify a number of requirements of a control system for an intelligent autonomous mobile robot. They each put constraints on possible control systems that we may employ. They are identified as follows.

Multiple Goals: Often the robot will have multiple goals, some conflicting, which it is trying to achieve. It may be trying to reach a certain point ahead of it while avoiding local obstacles. It may be trying to reach a certain place in minimal time while conserving power reserves. Often the relative importance of goals will be context-dependent. Getting off the railroad tracks when a train is heard becomes much more important than inspecting the last ten track ties of the current track section. The control system must be responsive to high priority goals, while still servicing necessary "low-level" goals (e.g., in getting off the railroad tracks, it is still important that the robot maintains its balance so it doesn't fall down).

Multiple Sensors: The robot will most likely have multiple sensors (e.g., TV cameras, encoders on steering and drive mechanisms, infrared beacon detectors, an inertial navigation

0882-4967/86/0300-0014$01.00

system, acoustic rangefinders, infrared rangefinders, access to a global positioning satellite system, etc.). All sensors have an error component in their readings. Furthermore, often there is no direct analytic mapping from sensor values to desired physical quantities. Some of the sensors will overlap in the physical quantities they measure. They will often give inconsistent readings—sometimes due to normal sensor error and sometimes due to the measurement conditions being such that the sensor (and subsequent processing) is used outside its domain of applicability. Often there will be no analytic characterization of the domain of applicability (e.g. under what precise conditions does the Sobel operator return valid edges?). The robot must make decisions under these conditions.

Robustness: The robot ought to be robust. When some sensors fail it should be able to adapt and cope by relying on those still functional. When the environment changes drastically it should be able to still achieve some modicum of sensible behavior, rather then sit in shock or wander aimlessly and irrationally around. Ideally it should also continue to function well when there are faults in parts of its processor(s).

Extensibility: As more sensors and capabilities are added to a robot it needs more processing power; otherwise, the original capabilities of the robot will be impaired relative to the flow of time.

B. Other Approaches

Multiple Goals: Elfes and Talukdar [4] designed a control language for Moravec's robot [11], which tried to accommodate multiple goals. It mainly achieved this by letting the user explicitly code for parallelism and to code an exception path to a special handler for each plausible case of unexpected conditions.

Multiple Sensors: Flynn [5] explicitly investigated the use of multiple sensors, with complementary characteristics (sonar is wide angle but reasonably accurate in depth, while infrared is very accurate in angular resolution but terrible in depth measurement). Her system has the virtue that if one sensor fails the other still delivers readings that are useful to the higher level processing. Giralt *et al.* [6] use a laser range finder for map making, sonar sensors for local obstacle detection, and infrared beacons for map calibration. The robot operates in a mode in which one particular sensor type is used at a time and the others are completely ignored, even though they may be functional. In the natural world multiple redundant sensors are abundant. For instance [10] reports that pigeons have more than four independent orientation sensing systems (e.g., sun position compared to internal biological clock). It is interesting that the sensors do not seem to be combined but rather, depending on the environmental conditions and operational level of sensor subsystems, the data from one sensor tends to dominate.

Robustness: The above work tries to make systems robust in terms of sensor availability, but little has been done with making either the behavior or the processor of a robot robust.

Extensibility: There are three ways this can be achieved without completely rebuilding the physical control system. 1) Excess processor power that was previously being wasted can be utilized. Clearly this is a bounded resource. 2) The processor(s) can be upgraded to an architecturally compatible but faster system. The original software can continue to run, but now excess capacity will be available and we can proceed as in the first case. 3) More processors can be added to carry the new load. Typically systems builders then get enmeshed in details of how to make all memory uniformly accessible to all processors. Usually the cost of the memory to processor routing system soon comes to dominate the cost (the measure of cost is not important—it can be monetary, silicon area, access time delays, or something else) of the system. As a result there is usually a fairly small upper bound (on the order of hundreds for traditional style processing units; on the order to tens to hundreds of thousands for extremely simple processors) on the number of processors which can be added.

C. Starting Assumptions

Our design decisions for our mobile robot are based on the following nine dogmatic principles (six of these principles were presented more fully in [2]).

1) Complex (and useful) behavior need not necessarily be a product of an extremely complex control system. Rather, complex behavior may simply be the reflection of a complex environment [13]. It may be an observer who ascribes complexity to an organism—not necessarily its designer.

2) Things should be simple. This has two applications. a) When building a system of many parts one must pay attention to the interfaces. If you notice that a particular interface is starting to rival in complexity the components it connects, then either the interface needs to be rethought or the decomposition of the system needs redoing. b) If a particular component or collection of components solves an unstable or ill-conditioned problem, or, more radically, if its design involved the solution of an unstable or ill-conditioned problem, then it is probably not a good solution from the standpoint of robustness of the system.

3) We want to build cheap robots that can wander around human-inhabited space with no human intervention, advice, or control and at the same time do useful work. Map making is therefore of crucial importance even when idealized blueprints of an environment are available.

4) The human world is three-dimensional; it is not just a two-dimensional surface map. The robot must model the world as three-dimensional if it is to be allowed to continue cohabitation with humans.

5) Absolute coordinate systems for a robot are the source of large cumulative errors. Relational maps are more useful to a mobile robot. This alters the design space for perception systems.

6) The worlds where mobile robots will do useful work are not constructed of exact simple polyhedra. While polyhedra may be useful models of a realistic world, it is a mistake to build a special world such that the models can be exact. For

this reason we will build no artificial environment for our robot.

7) Sonar data, while easy to collect, does not by itself lead to rich descriptions of the world useful for truly intelligent interactions. Visual data is much better for that purpose. Sonar data may be useful for low-level interactions such as real-time obstacle avoidance.

8) For robustness sake the robot must be able to perform when one or more of its sensors fails or starts giving erroneous readings. Recovery should be quick. This implies that built-in self calibration must be occurring at all times. If it is good enough to achieve our goals then it will necessarily be good enough to eliminate the need for external calibration steps. To force the issue we do not incorporate any explicit calibration steps for our robot. Rather we try to make all processing steps self calibrating.

9) We are interested in building *artificial beings*—robots that can survive for days, weeks and months, without human assistance, in a dynamic complex environment. Such robots must be self-sustaining.

II. Levels and Layers

There are many possible approaches to building an autonomous intelligent mobile robot. As with most engineering problems, they all start by decomposing the problem into pieces, solving the subproblems for each piece, and then composing the solutions. We think we have done the first of these three steps differently to other groups. The second and third steps also differ as a consequence.

A. Levels of Competence

Typically, mobile robot builders (e.g., [3], [6], [8], [11], [12], [14], [Tsuji 84], [Crowley 85]) have sliced the problem into some subset of

- sensing
- mapping sensor data into a world representation
- planning
- task execution
- motor control.

This decomposition can be regarded as a horizontal decomposition of the problem into vertical slices. The slices form a chain through which information flows from the robot's environment, via sensing, through the robot and back to the environment, via action, closing the feedback loop (of course most implementations of the above subproblems include internal feedback loops also). An instance of each piece must be built in order to run the robot at all. Later changes to a particular piece (to improve it or extend its functionality) must either be done in such a way that the interfaces to adjacent pieces do not change, or the effects of the change must be propagated to neighboring pieces, changing their functionality, too.

We have chosen instead to decompose the problem vertically as our primary way of slicing up the problem. Rather than slice the problem on the basis of internal workings of the solution, we slice the problem on the basis of desired external manifestations of the robot control system.

To this end we have defined a number of *levels of competence* for an autonomous mobile robot. A level of competence is an informal specification of a desired class of behaviors for a robot over all environments it will encounter. A higher level of competence implies a more specific desired class of behaviors.

We have used the following levels of competence (an earlier version of these was reported in [1]) as a guide in our work.

0) Avoid contact with objects (whether the objects move or are stationary).
1) Wander aimlessly around without hitting things.
2) "Explore" the world by seeing places in the distance that look reachable and heading for them.
3) Build a map of the environment and plan routes from one place to another.
4) Notice changes in the "static" environment.
5) Reason about the world in terms of identifiable objects and perform tasks related to certain objects.
6) Formulate and execute plans that involve changing the state of the world in some desirable way.
7) Reason about the behavior of objects in the world and modify plans accordingly.

Notice that each level of competence includes as a subset each earlier level of competence. Since a level of competence defines a class of valid behaviors it can be seen that higher levels of competence provide additional constraints on that class.

B. Layers of Control

The key idea of levels of competence is that we can build layers of a control system corresponding to each level of competence and simply add a new layer to an existing set to move to the next higher level of overall competence.

We start by building a complete robot control system that achieves level 0 competence. It is debugged thoroughly. We never alter that system. We call it the zeroth-level control system. Next we build a another control layer, which we call the first-level control system. It is able to examine data from the level 0 system and is also permitted to inject data into the internal interfaces of level 0 suppressing the normal data flow. This layer, with the aid of the zeroth, achieves level 1 competence. The zeroth layer continues to run unaware of the layer above it which sometimes interferes with its data paths.

The same process is repeated to achieve higher levels of competence (Fig. 3). We call this architecture a *subsumption architecture.*

In such a scheme we have a working control system for the robot very early in the piece—as soon as we have built the first layer. Additional layers can be added later, and the initial working system need never be changed.

We claim that this architecture naturally lends itself to solving the problems for mobile robots delineated in Section I-A.

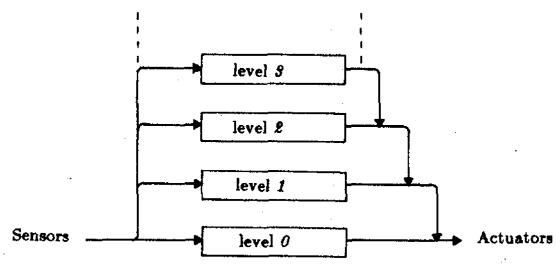


Fig. 3. Control is layered with higher level layers subsuming the roles of lower level layers when they wish to take control. The system can be partitioned at any level, and the layers below form a complete operational control system.

Multiple Goals: Individual layers can be working on individual goals concurrently. The suppression mechanism then mediates the actions that are taken. The advantage here is that there is no need to make an early decision on which goal should be pursued. The results of pursuing all of them to some level of conclusion can be used for the ultimate decision.

Multiple Sensors: In part we can ignore the sensor fusion problem as stated earlier using a subsumption architecture. Not all sensors need to feed into a central representation. Indeed, certain readings of all sensors need not feed into central representations—only those which perception processing identifies as extremely reliable might be eligible to enter such a central representation. At the same time however the sensor values may still be being used by the robot. Other layers may be processing them in some fashion and using the results to achieve their own goals, independent of how other layers may be scrutinizing them.

Robustness: Multiple sensors clearly add to the robustness of a system when their results can be used intelligently. There is another source of robustness in a subsumption architecture. Lower levels that have been well debugged continue to run when higher levels are added. Since a higher level can only suppress the outputs of lower levels by actively interfering with replacement data, in the cases that it can not produce results in a timely fashion the lower levels will still produce sensible results—albeit at a lower level of competence.

Extensibility: An obvious way to handle extensibility is to make each new layer run on its own processor. We will see below that this is practical as there are in general fairly low bandwidth requirements on communication channels between layers. In addition we will see that the individual layers can easily be spread over many loosely coupled processors.

C. Structure of Layers

But what about building each individual layer? Don't we need to decompose a single layer in the traditional manner? This is true to some extent, but the key difference is that we don't need to account for all desired perceptions and processing and generated behaviors in a single decomposition. We are free to use different decompositions for different sensor-set task-set pairs.

We have chosen to build layers from a set of small processors that send messages to each other. Each processor is a finite state machine with the ability to hold some data structures. Processors send messages over connecting "wires." There is no handshaking or acknowledgement of messages. The processors run completely asynchronously, monitoring their input wires, and sending messages on their output wires. It is possible for messages to get lost—it actually happens quite often. There is no other form of communication between processors, in particular there is no shared global memory.

All processors (which we refer to as modules) are created equal in the sense that within a layer there is no central control. Each module merely does its thing as best it can.

Inputs to modules can be suppressed and outputs can be inhibited by wires terminating from other modules. This is the mechanism by which higher level layers subsume the role of lower levels.

III. A Robot Control System Specification Language

There are two aspects to the components of our layered control architecture. One is the internal structure of the modules, and the second is the way in which they communicate. In this section we flesh out the details of the semantics of our modules and explain a description language for them.

A. Finite State Machines

Each module is a finite state machine, augmented with some instance variables, which can actually hold Lisp data structures.

Each module has a number of input lines and a number of output lines. Input lines have single-element buffers. The most recently arrived message is always available for inspection. Messages can be lost if a new one arrives on an input line before the last was inspected. There is a distinguished input to each module called *reset*. Each state is named. When the system first starts up all modules start in the distinguished state named NIL. When a signal is received on the reset line the module switches to state NIL. A state can be specified as one of four types.

Output	An output message, computed as a function of the module's input buffers and instance variables, is sent to an output line. A new specified state is then entered.
Side effect	One of the module's instance variables is set to a new value computed as a function of its input buffers and variables. A new specified state is then entered.
Conditional dispatch	A predicate on the module's instance variables and input buffers is computed and depending on the outcome one of two subsequent states is entered.

Event dispatch — A sequence of pairs of conditions and states to branch to are monitored until one of the events is true. The events are in combinations of arrivals of messages on input lines and the expiration of time delays.[1]

An example of a module defined in our specification language is the Avoid module in Listing 1.

Listing 1. Avoid module in Lisp.

```
(defmodule avoid 1
  :inputs (force heading)
  :outputs (command)
  :instance-vars (resultforce)
  :states
    ((nil (event-dispatch (and force heading) plan))
    (plan (setf resultforce (select-direction force heading))
          go)
    (go (conditional-dispatch (significant-force-p resultforce 1.0)
                              start
                              nil))
    (start (output command (follow-force resultforce))
           nil)))
```

Here, select-direction, significant-force-p, and follow-force are all Lisp functions, while setf is the modern Lisp assignment special form.

The force input line inputs a force with magnitude and direction found by treating each point found by the sonars as the site of a repulsive force decaying as the square of distance. Function select-direction takes this and combines it with the input on the heading line considered as a motive force. It selects the instantaneous direction of travel by summing the forces acting on the robot. (This simple technique computes the tangent to the minimum energy path computed by [9].)

The function significant-force-p checks whether the resulting force is above some threshold—in this case it determines whether the resulting motion would take less than a second. The dispatch logic then ignores such motions. The function follow-force converts the desired direction and force magnitude into motor velocity commands.

This particular module is part of the level 1 control system (as indicated by the argument "1" following avoid, the name of the module), which is described in Section IV-B. It essentially does local navigation, making sure obstacles are avoided by diverting a desired heading away from obstacles. It does not deliver the robot to a desired location—that is the task of level 2 competence.

B. Communication

Fig. 4 shows the best way to think about these finite state modules for the purposes of communications. They have some input lines and some output lines. An output line from one module is connected to input lines of one or more other modules. One can think of these lines as wires, each with sources and a destination. Additionally, outputs may be inhibited, and inputs may be suppressed.

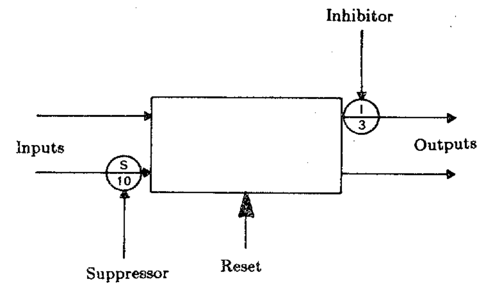


Fig. 4. A module has input and output lines. Input signals can be suppressed and replaced with the suppressing signal. Output signals can be inhibited. A module can also be reset to state NIL.

An extra wire can terminate (i.e. have its destination) at an output site of a module. If any signal travels along this wire it *inhibits* any output message from the module along that line for some predetermined time. Any messages sent by the module to that output during that time period is lost.

Similarly, an extra wire can terminate at an input site of a module. Its action is very similar to that of inhibition, but additionally, the signal on this wire, besides inhibiting signals along the usual path, actually gets fed through as the input to the module. Thus it suppresses the usual input and provides a replacement. If more than one suppressing wire is present they are essentially OR-ed together. For both suppression and inhibition we write the time constants inside the circle.

In our specification language we write wires as a source (i.e. an output line) followed by a number of destinations (i.e. input lines). For instance the connection to the force input of the Avoid module might be the wire defined as

(defwire 1 (feelforce force) (avoid force)).

This links the force output of the Feelforce module to the input of the Avoid module in the level one control system.

Suppression and inhibition can also be described with a small extension to the syntax above. Below we see the suppression of the command input of the Turn module, a level 0 module by a signal from the level 1 module Avoid.

(defwire 1 (avoid command) ((suppress (turn command) 20.0))).

In a similar manner a signal can be connected to the reset input of a module.

IV. A Robot Control System Instance

We have implemented a mobile robot control system to achieve levels 0 and 1 competence as defined above, and have started implementation of level 2 bringing it to a stage which exercises the fundamental subsumption idea effectively. We need more work on an early vision algorithm to complete level 2.

A. Zeroth Level

The lowest level layer of control makes sure that the robot does not come into contact with other objects. It thus achieves level 0 competence (Fig. 5). If something approaches the robot it will move away. If in the course of moving itself it is about

[1] The exact semantics are as follows. After an event dispatch is executed all input lines are monitored for message arrivals. When the next event dispatch is executed it has access to latches which indicate whether new messages arrived on each input line. Each condition is evaluated in turn. If it is true then the dispatch to the new state happens. Each condition is an and/or expression on the input line latches. In addition, condition expressions can include delay terms, which become true a specified amount of time after the beginning of the execution of the event dispatch. An event dispatch waits until one of its condition expressions is true.

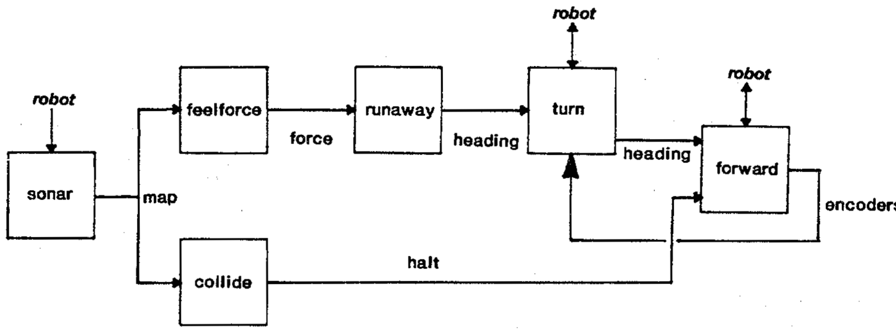


Fig. 5. Level 0 control system.

to collide with an object it will halt. Together these two tactics are sufficient for the robot to flee from moving obstacles, perhaps requiring many motions, without colliding with stationary obstacles. The combination of the tactics allows the robot to operate with very coarsely calibrated sonars and a wide range of repulsive force functions. Theoretically, the robot is not invincible of course, and a sufficiently fast-moving object or a very cluttered environment might result in a collision. Over the course of a number of hours of autonomous operation, our physical robot (see Section V-B) has not collided with either a moving or fixed obstacle. The moving obstacles have, however, been careful to move slowly.

The Turn and Forward modules communicate with the actual robot. They have extra communication mechanisms, allowing them to send and receive commands to and from the physical robot directly. The Turn module receives a heading specifying an in-place turn angle followed by a forward motion of a specified magnitude. It commands the robot to turn (and at the same time sends a busy message on an additional output channel described in Fig. 7) and on completion passes on the heading to the Forward module (and also reports the shaft encoder readings on another output line shown in Fig. 7). The Turn module then goes into a wait state ignoring all incoming messages. The Forward module commands the robot to move forward but halts the robot if it receives a message on its halt input line during the motion. As soon as the robot is idle, it sends out the shaft encoder readings. The message acts as a reset for the Turn module, which is then once again ready to accept a new motion command. Notice the any heading commands sent to the Turn module during transit are lost.

The Sonar module takes a vector of sonar readings, filters them for invalid readings, and effectively produces a robot centered map of obstacles in polar coordinates.

The Collide module monitors the sonar map and if it detects objects dead ahead, it sends a signal on the halt line to the Motor module. The Collide module does not know or care whether the robot is moving. Halt messages sent while the robot is stationary are essentially lost.

The Feelforce module sums the results of considering each detected object as a repulsive force, generating a single resultant force.

The Runaway module monitors the 'force' produced by the sonar detected obstacles and sends commands to the turn module if it ever becomes significant.

Fig. 5 gives a complete description of how the modules are connected together.

B. First Level

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 was defined earlier as level 1 competence. This control level relies in a large degree on the zeroth level's aversion to hitting obstacles. In addition it uses a simple heuristic to plan ahead a little in order to avoid potential collisions which would need to be handled by the zeroth level.

The Wander module generates a new heading for the robot every ten seconds or so.

The Avoid module, described in more detail in Section III, takes the result of the force computation from the zeroth level and combines it with the desired heading to produce a modified heading, which usually points in roughly the right direction, but is perturbed to avoid any obvious obstacles. This computation implicitly subsumes the computations of the Runaway module, in the case that there is also a heading to consider. In fact the output of the Avoid module suppresses the output from the Runaway module as it enters the Motor module.

Fig. 6 gives a complete description of how the modules are connected together. Note that it is simply Fig. 5 with some more modules and wires added.

C. Second Level

Level 2 is meant to add an exploratory mode of behavior to the robot, using visual observations to select interesting places to visit. A vision module finds corridors of free space. Additional modules provide a means of position servoing the robot to along the corridor despite the presence of local obstacles on its path (as detected with the sonar sensing system). The wiring diagram is shown in Fig. 7. Note that it is simply Fig. 6 with some more modules and wires added.

The Status module monitors the Turn and Forward modules. It maintains one status output which sends either hi or lo messages to indicate whether the robot is busy. In addition, at

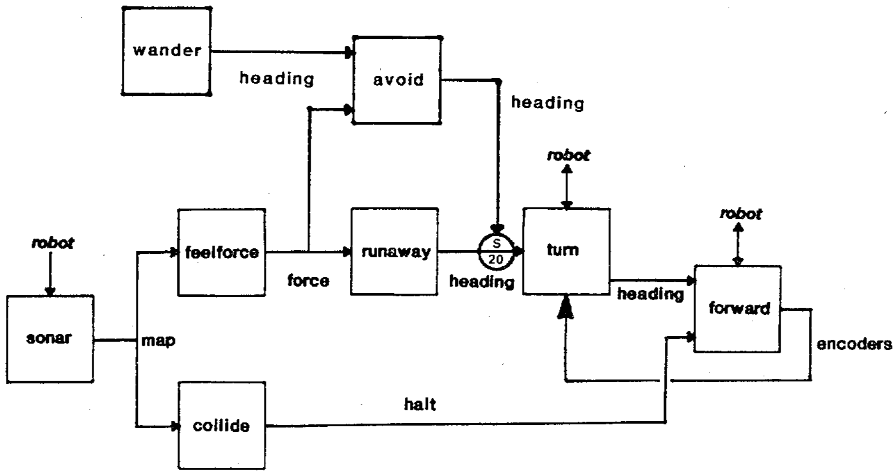


Fig. 6. Level 0 control system augmented with the level 1 system.

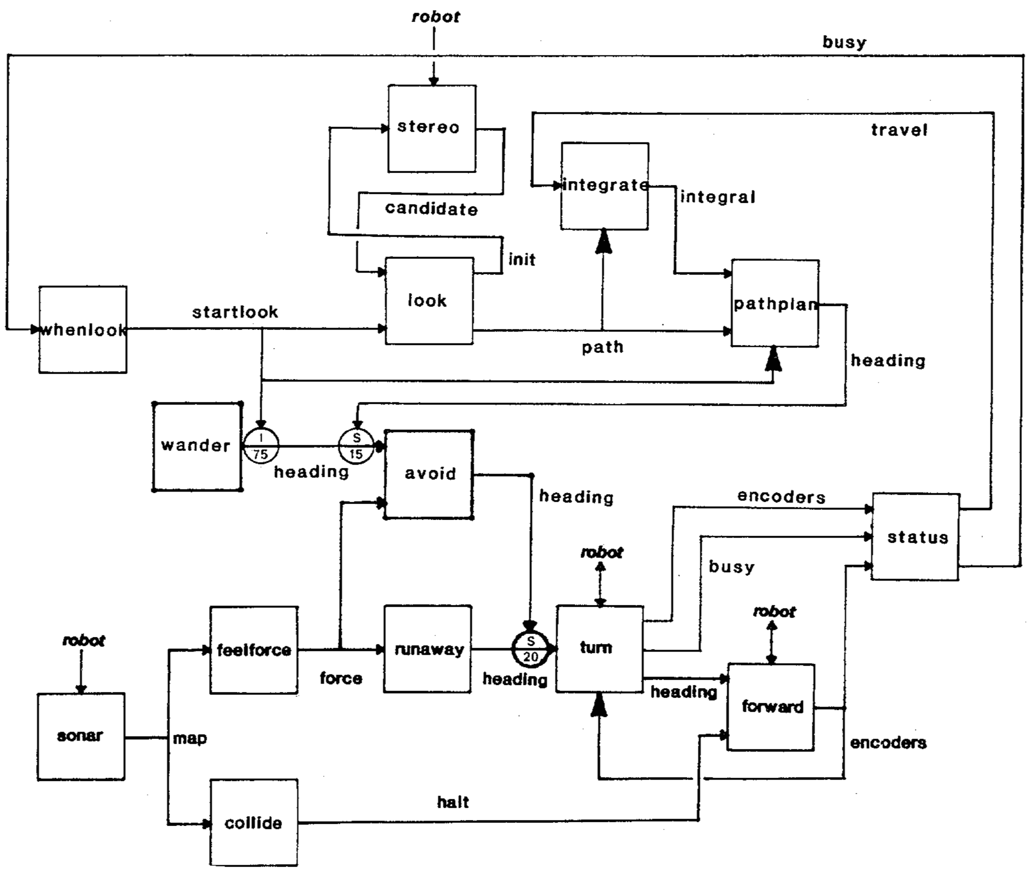


Fig. 7. Level 0 and 1 control systems augmented with the level 2 system.

the completion of every turn and roll forward combination it sends out a combined set of shaft encoder readings.

The Whenlook module monitors the busy line from the Status module, and whenever the robot has been sitting idle for a few seconds it decides its time to look for a corridor to traverse. It inhibits wandering so it can take some pictures and process them without wandering away from its current location, and resets the Pathplan and Integrate modules. This latter action ensures that the robot will know how far it has moved from its observation point should any Runaway impulses perturb it.

The Look module initiates the vision processing, and waits for a candidate freeway. It filters out poor candidates and passes any acceptable one to the Pathplan module.

The Stereo module is supposed to use stereo TV images [7], which are obtained by the robot, to find a corridor of free space. At the time of writing final version of this module had not been implemented. Instead, both in simulation and on the physical robot, we have replaced it with a sonar-base corridor finder.

The Integrate module accumulates reports of motions from the status module and always sends its most recent result out on its integral line. It gets restarted by application of a signal to its reset input.

The Pathplan module takes a goal specification (in terms of an angle to turn, a distance to travel) and attempts to reach that goal. To do this, it sends headings to the Avoid module, which may perturb them to avoid local obstacles, and monitors its integral input which is an integration of actual motions. The messages to the Avoid module suppress random wanderings of the robot, so long as the higher level planner remains active. 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) it terminates.

The current wiring of the second level of control is shown in Fig. 7, which augments the two lower level control systems. The zeroth and first layers still play an active roll during normal operation of the second layer.

V. Performance

The control system described here has been used extensively to control both a simulated robot and an actual physical robot wandering around a cluttered laboratory and a machine room.

A. A Simulated Robot

The simulation tries to simulate all the errors and uncertainties that exist in the world of the real robot. When commanded to turn through angle α and travel distance d the simulated robot actually turns through angle $\alpha + \delta\alpha$ and travels distance $d + \delta d$. Its sonars can bounce off walls multiple times, and even when they do return they have a noise component in the readings that model thermal and humidity effects. We feel it is important to have such a realistic simulation. Anything less leads to incorrect control algorithms.

The simulator runs off a clock and runs at the same rate as would the actual robot. It actually runs on the same processor that is simulating the subsumption architecture. Together they are nevertheless able to perform a real-time simulation of the robot and its control and also drive graphics displays of robot state and module performance monitors. Fig. 8 shows the robot (which itself is not drawn) receiving sonar reflections at some of its 12 sensors. Other beams did not return within the time allocated for data collection. The beams are being reflected by various walls. There is a small bar in front of the robot perpendicular to the direction the robot is pointing.

Fig. 9 shows an example world in two dimensional projection. The simulated robot with a first level control system connected was allowed to wander from an initial position. The squiggly line traces out its path. Note that it was wandering aimlessly and that it hit no obstacles.

Fig. 10 shows two examples of the same scene and the motion of the robot with the second level control system connected. In these cases the Stereo module was supplanted with a situation-specific module, which gave out two precise corridor descriptions. While achieving the goals of following these corridors the lower level wandering behavior was suppressed. However the obstacle avoiding behavior of the lower levels continued to function—in both cases the robot avoided the square obstacle. The goals were not reached exactly. The simulator models a uniformly distributed error of ± 5 percent in both turn and forward motion. As soon as the goals had been achieved satisfactorily the robot reverted to its wandering behavior.

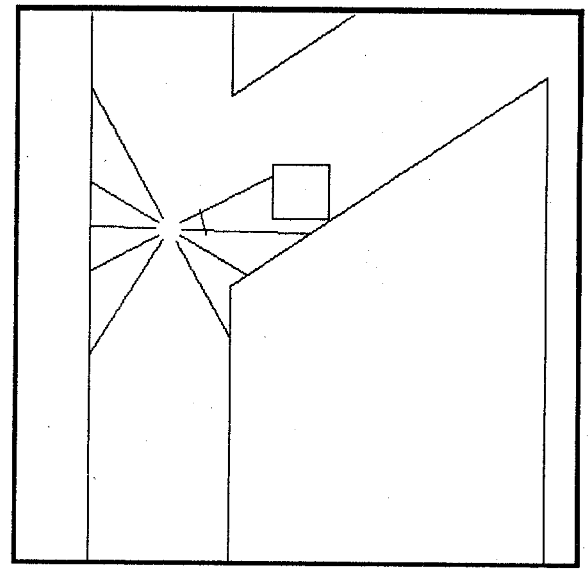


Fig. 8. Simulated robot receives 12 sonar readings. Some sonar beams glance off walls and do not return within a certain time.

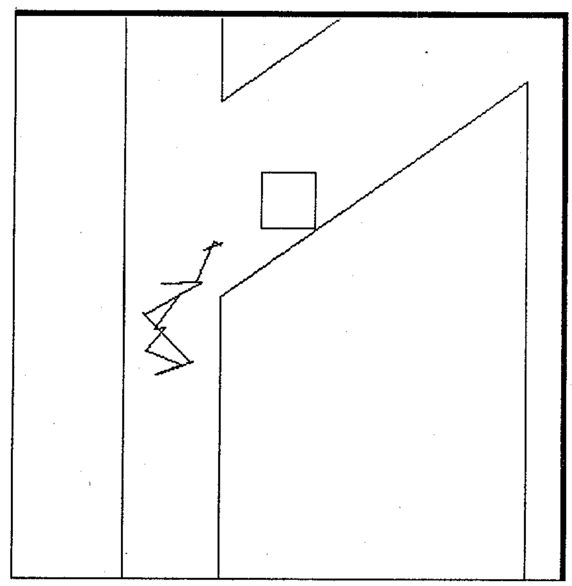
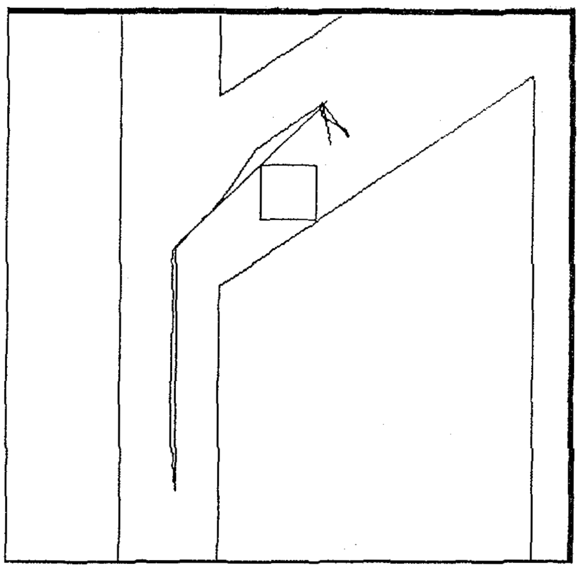


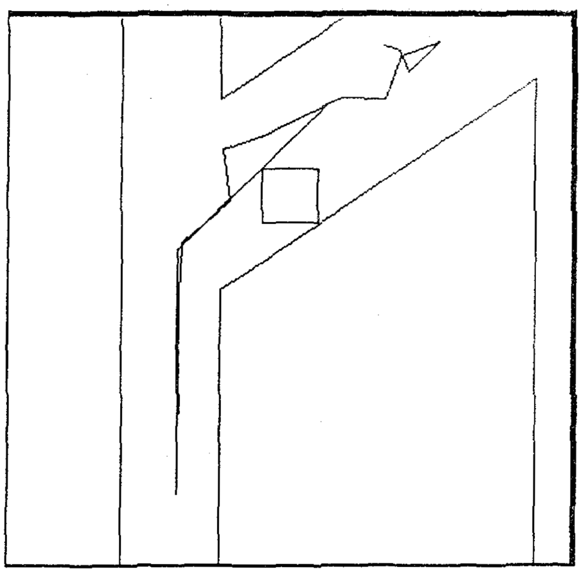
Fig. 9. Under levels 0 and 1 control the robot wanders around aimlessly. It does not hit obstacles.

B. A Physical Robot

We have constructed a mobile robot shown in Fig. 11. It is about 17 inches in diameter and about 30 inches from the ground to the top platform. Most of the processing occurs offboard on a Lisp machine.



(a)



(b)

Fig. 10. (a) With level 2 control the robot tries to achieve commanded goals. The nominal goals are the two straight lines. (b) After reaching the second goal, since there are no new goals forthcoming, the robot reverts to aimless level 1 behavior.

The drive mechanism was purchased from Real World Interface of Sudbury, MA. Three parallel drive wheels are steered together. The two motors are servoed by a single microprocessor. The robot body is attached to the steering mechanism and always points in the same direction as the wheels. It can turn in place (actually it inscribes a circle about 1 cm in diameter).

Currently installed sensors are a ring of twelve Polaroid sonar time-of-flight range sensors and two Sony CCD cameras. The sonars are arranged symmetrically around the rotating body of the robot. The cameras are on a tilt head (pan is provided by the steering motors). We plan to install feelers that can sense objects at ground level about six inches from the base extremities.

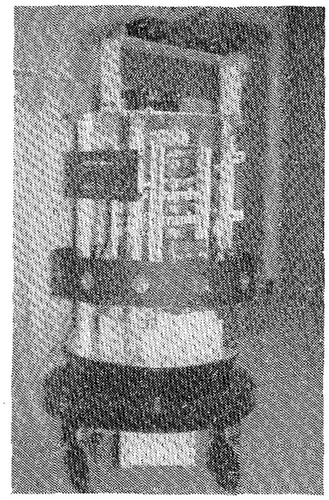


Fig. 11. The M.I.T. AI Lab mobile robot.

A central cardcage contains the main on-board processor, an Intel 8031. It communicates with off-board processors via a 12 Kbit/s duplex radio link. The radios are modified Motorola digital voice encryption units. Error correction cuts the effective bit rate to less than half the nominal rating. The 8031 passes commands down to the motor controller processor and returns encoder readings. It controls the sonars and the tilt head, and it switches the cameras through a single channel video transmitter mounted on top of the robot. The latter transmits a standard TV signal to a Lisp machine equipped with a demodulator and frame grabber.

The robot has spent a few hours wandering around a laboratory and a machine room.

Under level 0 control, the robot finds a large empty space and then sits there contented until a moving obstacle approaches. Two people together can successfully herd the robot just about anywhere—through doors or between rows of disk drives, for instance.

When level 1 control is added the robot is no longer content to sit in an open space. After a few seconds it heads off in a random direction. Our uncalibrated sonars and obstacle repulsion functions make it overshoot a little to locations where the Runaway module reacts. It would be interesting to make this the basis of adaption of certain parameters.

Under level 2 a sonar-based corridor finder usually finds the most distant point in the room. The robot heads of in the direction. People walking in front of the robot cause it to detour, but the robot still gets to the initially desired goal, even when it involves squeezing between closely spaced obstacles. If the sonars are in error and a goal is selected beyond a wall, the robot usually ends up in a position where the attractive force of the goal is within a threshold used by Avoid of the repulsive forces of the wall. At this point Avoid does not issue any heading, as it would be for some trivial motion of the robot. The robot sits still defeated by the obstacle. The Whenlook module, however, notices that the robot is idle and initiates a new scan for another corridor of free space to follow.

C. Implementation Issues

While we have been able to simulate sufficient processors on a single Lisp machine up until now, that capability will soon pass as we bring on line our vision work (the algorithms have been debugged as traditional serial algorithms, but we plan on re-implementing them within the subsumption architecture). Building the architecture in custom chips is a long-term goal.

One of the motivations for developing the layered control system was extensibility of processing power. The fact that it is decomposed into asynchronous processors with low-bandwidth communication and no shared memory should certainly assist in achieving that goal. New processors can simply be added to the network by connecting their inputs and outputs at appropriate places—there are no bandwidth or synchronization considerations in such connections.

The finite state processors need not be large. Sixteen states is more than sufficient for all modules we have written so far. (Actually, eight states are sufficient under the model of the processors we have presented here and used in our simulations. However we have refined the design somewhat towards gate-level implementation, and there we use simpler more numerous states.) Many such processors could easily be packed on a single chip.

The Lisp programs that are called by the finite state machines are all rather simple. We believe it is possible to implement each of them with a simple network of comparators, selectors, polar coordinate vector adders, and monotonic function generators. The silicon area overhead for each module would probably not be larger than that required for the finite state machine itself.

VI. Conclusion

The key ideas in this paper are the following.

1) The mobile robot control problem can be decomposed in terms of behaviors rather than in terms of functional modules.
2) It provides a way to incrementally build and test a complex mobile robot control system.
3) Useful parallel computation can be performed on a low bandwidth loosely coupled network of asynchronous simple processors. The topology of that network is relatively fixed.
4) There is no need for a central control module of a mobile robot. The control system can be viewed as a system of agents each busy with their own solipsist world.

Besides leading to a different implementation strategy it is also interesting to note the way the decomposition affected the capabilities of the robot control system we have built. In particular, our control system deals with moving objects in the environment at the very lowest level, and it has a specific module (Runaway) for that purpose. Traditionally mobile robot projects have delayed handling moving objects in the environment beyond the scientific life of the project.

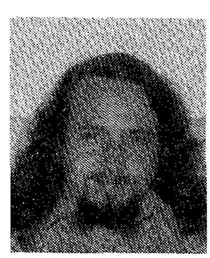
Note: A drawback of the presentation in this paper was merging the algorithms for control of the robot with the implementation medium. We felt this was necessary to convince the reader of the utility of both. It is unlikely that the subsumption architecture would appear to be useful without a clear demonstration of how a respectable and useful algorithm can run on it. Mixing the two descriptions as we have done demonstrates the proposition.

Acknowledgment

Tomás Lozano-Pérez, Eric Grimson, Jon Connell, and Anita Flynn have all provided helpful comments on earlier drafts of this paper.

References

[1] R. A. Brooks, "Aspects of mobile robot visual map making," in *Robotics Research 2,* Hanafusa and Inoue, Eds. Cambridge, MA: M.I.T., 1984, pp. 369–375.
[2] ——, "Visual map making for a mobile robot," in *Proc. 1985 IEEE Conf. Robotics and Automat.,* pp. 824–829.
[3] James L. Crowley, "Navigation for an intelligent mobile robot," *IEEE J. Robotics Automat.,* vol. RA-1, no. 1, Mar. 1985, pp. 31–41.
[4] A. Elfes and S. N. Talukdar, "A distributed control system for the CMU rover," in *Proc. IJCAI,* 1983, pp. 830–833.
[5] A. Flynn, "Redundant sensors for mobile robot navigation," M.S. Thesis, Department of Electrical Engineering and Computer Science, M.I.T., Cambridge, MA, July 1985.
[6] G. Giralt, R. Chatila, and M. Vaisset, "An integrated navigation and motion control system for autonomous multisensory mobile robots," in *Robotics Research 1,* Brady and Paul, Eds. Cambridge, MA: M.I.T. 1983, 191–214.
[7] W. L. Grimson, "Computational experiments with a feature based stereo algorithm," *IEEE Trans. Patt. Anal. Mach. Intell.,* vol. PAMI-7, pp. 17–34, Jan. 1985.
[8] Y. Kanayama, "Concurrent programming of intelligent robots," in *Proc. IJCAI,* 1983, pp. 834–838.
[9] O. Khatib, "Dynamic control of manipulators in operational space," *Sixth IFTOMM Cong. Theory of Machines and Mechanisms,* Dec. 1983.
[10] M. L. Kreithen, "Orientational strategies in birds: a tribute to W. T. Keeton," in *Behavioral Energetics: The Cost of Survival in Vertebrates.* Columbus, OH: Ohio State University, 1983, pp. 3–28.
[11] H. P. Moravec, "The stanford cart and the CMU rover," *Proc. IEEE,* vol. 71, pp. 872–884, July 1983.
[12] N. J. Nilsson, "Shakey the robot," SRI AI Center, tech. note 323, Apr. 1984.
[13] H. A. Simon, *Sciences of the Artificial.* Cambridge, MA: M.I.T., 1969.
[14] S. Tsuji, "Monitoring of a building environment by a mobile robot," in *Robotics Research 2,* Hanafusa and Inoue, Eds. Cambridge, MA: M.I.T., 1985, pp. 349–356.



Rodney A. Brooks was born in Adelaide, South Australia, on December 30, 1954. He received the B.Sc. and M.Sc. degrees in mathematics from the Flinders University of South Australia and the Ph.D. degree in computer science from Stanford University, Stanford, CA, in 1981.

Since 1981 he has held research positions at Carnegie-Mellon University, Pittsburgh, PA, and the Massachusetts Institute of Technology (M.I.T.), Cambridge, and faculty positions at Stanford and M.I.T. He is currently an Assistant Professor in the Department of Electrical Engineering and Computer Science at M.I.T., and a member of the Artificial Intelligence Laboratory. His research interests include computer vision, geometric reasoning, robot planning, and Lisp systems and compilers. His books include *Model-Based Computer Vision* from UMI Research Press and *Programming in Common Lisp* from John Wiley and Sons.