A Mobile Robot with Onboard Parallel Processor and Large Workspace Arm

Rodney A. Brooks, Jon Connell, and Anita Flynn

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

ABSTRACT

The MIT AI Lab's second mobile robot, MOBOT-2, has a number of unique design features. In this paper we describe two of them in detail. First, MOBOT-2 has an extremely cheap 32 processor distributed control system. The processor system, called BARNACLE, runs asynchronously with no central locus of control. Unlike almost all other parallel processors this one has no expensive communications routing network. The communication topology is determined by a distributed patch panel. All computing is done onboard the robot. Second, MOBOT-2 has an onboard arm. It is lightweight, but has an extremely large working volume. The arm is controlled by the parallel processor.

1. Introduction

The MIT AI Lab MOBOT project has been underway since January 1985. In that time we have built our first robot, MOBOT-1, and tested it wandering around laboratories and machine rooms [Brooks 86]. MOBOT-1 has 12 sonar depth sensors and a pair of TV cameras. It has an onboard microprocessor which communicates over a radio and TV link to an offboard Lisp machine where the real computing is done. It has no actuators other than its wheels. The main research emphasis with this first robot has been on a distributed parallel control system which is simulated on the Lisp machine. [Brooks 86] describes the motivations and strengths of the control architecture, known as the *subsumption* architecture.

Our second robot, the one described here, is intended to be an improvement on the first and to make for a richer experimental testbed for further pursuing the distributed control system. To this end it includes an onboard parallel processor which runs the subsumption architecture, and a lightweight arm which will enable the robot to do much more interesting tasks in the world than simply move around. The remainder of this paper explores the design decisions and trade-offs in these two aspects of MOBOT-2. Figure 1 shows MOBOT-1 and the new MOBOT-2.

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.

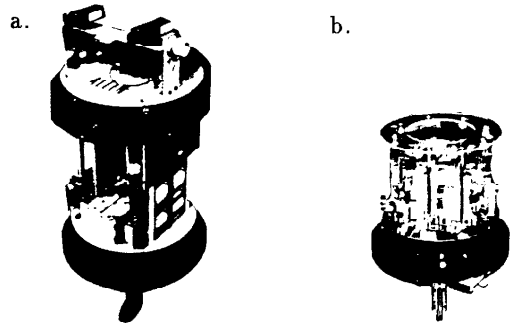


Figure 1. MIT mobile robots a. MOBOT-1 b. Partially constructed MOBOT-2.

1.1 Previous Lessons

We have found from previous experience with MOBOT-1 that we spend almost as much time taking the robot apart and putting it back together in order to enhance, modify or debug it, as we do actually running it. This is because of the fact that once it works at any given level, it becomes uninteresting and there's always something to add to make it more interesting. Hence, we would like MOBOT-2 to be extremely easy to strip down and reassemble. This dictates the type of physical fastening systems we use.

Another observation made from previous experience with mobile robots is that there is an explosive phenomenon regarding power and weight. Big robots need hefty motors, which call for large batteries. Large batteries add more weight, requiring larger motors and even bigger batteries (e.g. [Giralt et al 84]) and so on. We would like to reverse this trend and build each succeeding mobot smaller and lighter. In the limit, all our problems will be solved.

2. The Parallel Processor

All serious previous mobile robot projects (e.g. [Crowley 85], [Giralt et al 84], [Moravec 83], [Nilsson 84]) have used an offboard processor to do the bulk of the computation for perception, world modelling and planning as required by the robot. We adopted this approach for MOBOT-1. Now however, due to the availability of more computationally powerful, low power CMOS processors and a new decompostion of robot control systems [Brooks 86], we believe the time is ripe to move to all onboard processing.

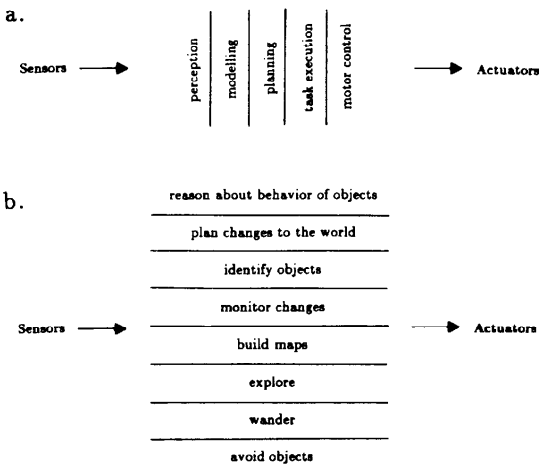


Figure 2. Slicing a control system a. Traditional decomposition b. Subsumption architecture

2.1 The Subsumption Architecture

The usual approach to building control systems for mobile robots is to decompose the problem into a series (roughly) of *functional units* as illustrated by a series of vertical slices in figure 2a. 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 figure 2b. As with a functional decomposition we implement each slice explicitly, then tie them together to form a robot control system. The difference is that after building the lowest layer we already have a control system which achieves a certain level of competence. We leave that running system intact and build a second layer to augment it. The process continues building layer upon layer to give successively more competent control systems as in figure 3.

We call this architecture a *subsumption architecture*. Our new decomposition leads to a radically different architecture for mobile robot control systems, with radically different implementation strategies plausible at the hardware level. It also confers a large number of advantages concerning robustness, buildability, and testability.

One of these advantages is that the architecture is never bandwidth limited. The most expensive part of most parallel processors is the switch which lets processors talk to each other or talk to memory. Under the subsumption architecture the topology of communications between processors is fixed for a particular run of the robot. Thus there is no need for a fast dynamic switch to reconfigure message topology, as this can essentially happen offline. Instead the communication topology is predetermined by a distributed patch panel.

2.2 Physical Layout

We have chosen to build layers from collections of very simple processor pairs, which we call modules. Each processor

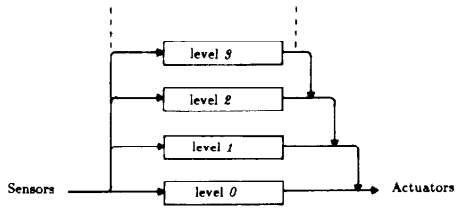


Figure 3. Augmenting an existing control system by adding more levels.

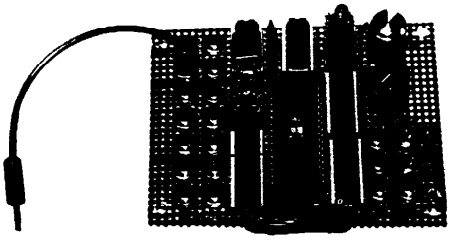


Figure 4. A processor board from BARNACLE.

pair consists of a finite state machine control processor and a peripheral geometry coprocessor. The finite state machine controls the data flow through the module. It can wait for certain inputs, save partial results in an internal register, or act as a watchdog timer. The finite state machine also occasionally passes data to its attached geometry coprocessor which computes functions such as polar coordinate vector addition, scaled comparison, and monotonic functions.

The current design calls for 32 processor boards. That is the number of modules required to control the robot arm and base. Our initial idea was that precisely one physical processor should play the role of one finite state machine-geometric coprocessor pair. Recently we are of the opinion that we may be able to simulate more than one processor pair per physical processor.

Each processor board is 4 inches wide by 3 inches high and contains a Hitachi 6301 microcontroller and logic for performing suppression (described below). The 6301 is a CMOS processor with an architecture similar to the 6800. It has 128 bytes of internal RAM and an external 8K piggy-back EPROM. The board shown in figure 4 also has provisions for adding an extra 2K of external RAM if needed. The EPROM will contain the program to emulate the finite state machine(s) and the geometric co-processor(s), as well as the code for communicating between modules. The RAM will hold the internal state variables of each module and serve as a scratch pad for the geometric processor and for the decoding of messages.

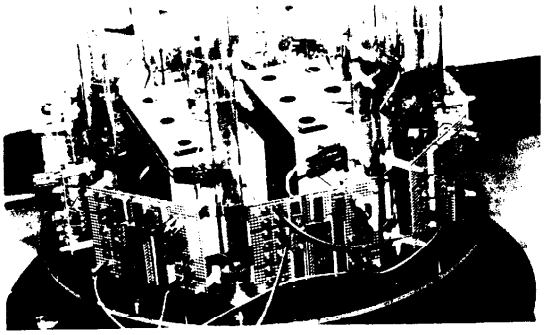


Figure 5. Physical arrangement of processor boards.

Each processor board has 3 serial inputs and 3 serial outputs. The input and output lines of the module terminate in subminature phone jacks on each individual board. Outputs of one module are connected to inputs of another using patch wires made from a piece of coaxial cable with plugs at each end. A single lead contains two conductors, one to carry data and the other to carry control signals. All input jacks come in pairs so that signals which must fan out can be daisy chained together. In addition each jack has a built in switch which forces the input to a known state if no jack is inserted.

Figure 5 illustrates the complete BARNACLE system. The processor boards are mounted around the periphery of the robot with the chips pointing outwards. Currently we plan on building four octagons and stacking them (conceptually) above each other on a plexiglass frame. Particular topologies for the processors and suppressor nodes will be patched together with the cables being clipped into these trays. The batteries required for running the robot reside in the center of the octagon.

2.3 Communication Between Modules

All messages in the BARNACLE processor are 24 bit packets. Making the packets a fixed size significantly simplifies the communication protocol. Our analysis showed that 24 bits was the right number for our application for several reasons. None of the physical quantities with which the robot must deal are known to more than 8 bits of accuracy nor can any of its effectors be controlled more precisely than this. Since 8 bits make a byte, that seemed like a reasonable size to represent any physical quantity. The remaining question was how many bytes per packet. The configuration of the robot in the world can be represented by three bytes: x, y, and θ. A motion command takes two bytes, one for heading, and one for distance. A third byte can represent speed if desired. A corridor description can fit in three bytes: length, width, and orientation relative to the current robot position. With 12 sonar sensors mounted on the robot (as in MOBOT-1, but more likely infrared depth sensors on MOBOT-2), two bits per ranger can be placed in a single 24 bit packet. Two bits is plenty for local obstacle detection and avoidance.

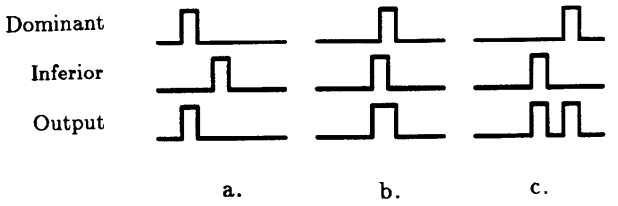


Figure 6. In each of the three cases above the dominant input suppresses the inferior input. a. Inferior packet is ignored. b. Inferior packet is blocked. c. Partially transmitted inferior packet is flushed.

Messages are sent serially over 2 conductors, one conductor is the data line and the other is a control line. Both control and data wires are connected to single bits of the processors parallel ports. We use 12 parallel I/O lines to supply three input and three output lines on each processor. All ports are polled every 500 microseconds (time for about 100 instructions) which is an easy speed to maintain. Packets of 24 bits can thus be sent in about one tenth of a second. This is much faster than the communication rate we used to run MOBOT-1 using our lisp machine simulation of BARNACLE.

Inhibition, a necessary part of the subsumption architecture, is accomplished in BARNACLE by simply forbidding the module to send any messages for a specified amount of time. There is a special inhibit input to each processors which is connected to a timer whose period can be varied from a tenth of a second to several minutes by a potentiometer. A pulse on the control line of the inhibit input starts the timer which in turn prevents the processor from sending any new messages, although it is allowed to finish ones currently in progress. The timer can be re-triggered at any time during the inhibition interval to extended the duration. When at last the timer's output goes low, the processor is free to send whatever messages it wishes.

Suppression in BARNACLE is also implemented using special hardware. Each processor board contains one suppressor node which has two inputs and one output. The state of a flip-flop controls which input makes it through to the output. This flip-flop is set by a pulse on the control line of the dominant input, and reset after a time-out period controlled by an associated potentiometer. Figure 6 shows how this works. The control line signals an impending message with a square pulse; the falling edge signals that the message is about to begin. There is a lower bound on the length of this pulse, but no upper bound. Reassertion of this line during a message indicates that the message is invalid and should be ignored–the new falling edge will indicate the start of a different message. Thus the two inputs to a suppressor node can be completely asynchronous and yet there are no timing or collision problems thanks to the protocol definition.

3. The Arm

The fact that our robot is mobile makes a big difference

in the design of the manipulator. A mobile robot is a lot like an airplane in that all its resources are severely limited. It can only carry a certain amount of *weight*, all its equipment must fit in a specified *size*, and the *power* available from batteries is limited. Not only is electrical power scarce, *computational power* is also scarce. Even with advanced VLSI and CMOS, the more processors there are and the faster they run, the more power, space, and weight they consume. Fortunately, unlike electrical power, it is possible to beam information into and out of a robot. Yet, because robots inhabit noisy environments full of fluorescent lights and disk drives, the telemetry bandwidth is limited and occasionally communications are interrupted altogether.

3.1 Special Requirements

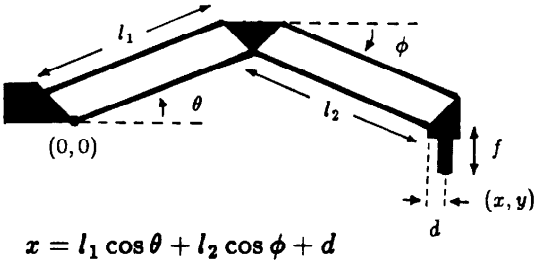
What has been said above is true for any piece of hardware residing on a mobile robot. There are, however, also specific requirements for manipulators on mobile robots. One of these is that the arm and its end effector must not be so heavy that it tips the robot over. This needs to be true of the manipulator throughout its workspace and range of payloads. We want our mobile to manipulate reasonably heavy objects. Arms that can lift a couple ounces are fine for shuffling semiconductor wafers, but they can't move coffee mugs or pick up rocks for examination. Making the actual arm light also means that it can move faster without encountering control instabilities.

Another shortcoming of most commercially available manipulators is that they have too small a workspace. Mobile robots inhabit a three dimensional world that has many different heights: desks, tables, shelves, floors, etc. This means that the manipulator's workspace needs a large amount of vertical freedom. The lateral mobility of the arm, however, does not need to be very big since the robot's base allows it to move the whole arm around.

Lastly, the precision of commercial manipulators is overkill for the actions we wish our robot to perform. We expect our mobots to transfer things from one location to another, not to do low tolerance assembly. Sensors such as vision can not locate an object to a thousand of an inch. On the other end, getting the gripper to within half an inch is sufficient to grasp most things and setting something down within an inch of where you want it is usually fine. For cases where absolute positioning does matter, like removing a peg from a hole, there are often environmental constraints (like the edges of the hole) that can aid in the alignment given an appropriate control system [Lozano-Pérez et al 83].

3.2 Mechanical Design

The mechanical design of the arm for MOBOT-2 is relatively straight forward. It is a 2 degree of freedom planar manipulator which moves in a vertically oriented plane passing through the central vertical axis of the robot. The two degrees of freedom are used to select a height for the gripper and to give fine grain control over the hand's radial position. Coarse radial and angular control is provided by moving the entire robot. To keep the number of joints down we have not provided fine grain control of the angular position of the hand; there is no sideways motion. The rationale behind this is that many long range sensors, in particular vision, supply very accurate headings toward a target object but relatively poor range estimates. Having a degree of freedom that can compensate for these errors is desirable. This is why with only 2 degrees of freedom we choose to mount the arm so it operates in a radial rather than a tangential plane.



$$x = l_1 \cos\theta + l_2 \cos\phi + d$$

$$y = l_1 \sin\theta + l_2 \sin\phi - f$$

Figure 7. The manipulator and its kinematics. Note that the hand is always vertical.

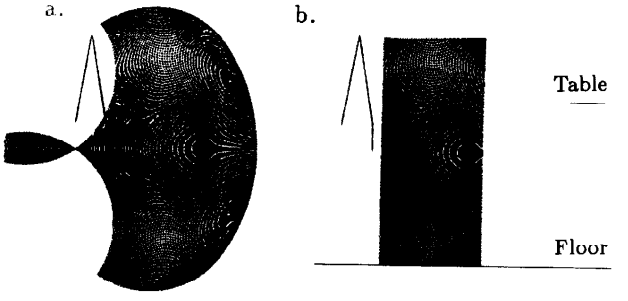


Figure 8. Tip positions for the manipulator. a. Total workspace. b. Normal operating area.

Each section of the arm consists of a parallel four-bar linkage. Figure 7 shows how these linkages are arranged. Because there is no wrist in this design we have decided to have the gripper always point straight down, an attitude which allows the hand to pick up small objects from flat surfaces. The four-bar mechanisms serve to reference the attitude of the gripper to the robot's frame. The motors which actuate the two joints are capable of lifting a payload of 2 pounds. Because the motors are light, each motor is located at the joint it controls. Mounting the motors back further would require a complicated power transmission system that could introduce an unacceptable phase lag in the servo control of the fingers and would likely weigh as much as the motor itself.

The complete workspace of the arm is shown in figure 8a.

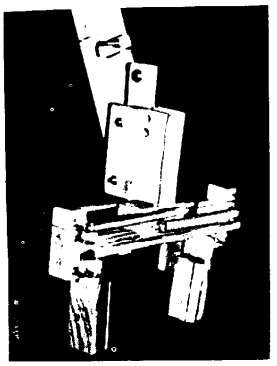


Figure 9. Close-up of a prototype compliant gripper.

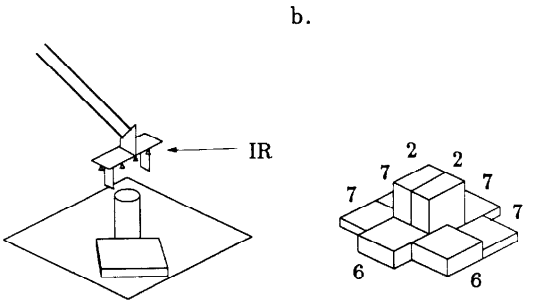


Figure 10. The hand is surrounded by a ring of IR range finders (triangles). a. The arm over a table. b. Heights returned by the IR ring.

However, we are primarily concerned with the vertical column shown in figure 8b which is 40 inches high and 18 inches wide. This allows the arm to work on both the floor and the tops of tables and to reach anywhere in the front half of a normal desk top. Quantizing the joint angles to 8 bits each gives the arm a quarter inch accuracy over the entire workspace. All 65536 possible fingertip positions are plotted in figure 8a.

The hand is a simple linear slide parallel jaw gripper. The fingers are 1 inch wide by 3 inches long and contact the object via two compliant rubber pads. Since there is no fine grain control over the angular location of the arm with respect to the robot, the jaws of gripper open to a wide 5 inches. This lateral leeway is important because it allows us to tolerate 2 degrees of error in the angular position of the arm at the furthest point in its workspace and larger errors at shorter extensions.

3.3 Sensors and Control

The arm is controlled by specifying a speed for each of the joints. This is accomplished by slowly ramping the control voltage to a standard proportional controller at a particular rate. Controlling the speed of the joints rather than their position lets us move the hand along a desired trajectory. In particular, we can command the arm to raise the gripper straight up or move it directly forward by specifying joint speeds that vary with the configuration of the arm.

Not only are the joint positions sensed, but the error voltages in the servo amps are also reported. If the servo has a known transfer function, such as a generalized spring, this can be very useful information. Errors in the joint angles indicate the amount of torque being supplied by the motors. By coupling this with the configuration of the arm we can determine the weight of the payload being carried. For the fingers, measuring the control error tells how tightly the hand is grasping an object. This allows us to close the fingers slowly until a sufficiently large grasping force is sensed. The amount of force necessary is determined by the weight of the payload which can either be measured directly as described above, or estimated by measuring the finger separation.

Aside from basic kinesthesic sensors, the hand also has a cluster of 8 infrared range-finders. These are arranged in a hexagonal grid surrounding the fingers as can be seen in figure 10a. Each sensor provides a coarse (3 bit) depth measurement of the surface in its field of view. For the table shown in figure 10a, the IR depth map looks figure 10b.

4. Conclusion

MOBOT-2 has an onboard parallel processor. The processor is unique in that it has no dynamic switch, but rather relies on physical configuration of its communication topology. There is no central locus of control in the entire system. The parallel processor controls motors on the onboard arm, reacts to local moving obstacles, processes sensor information, and formulates high level plans all in a distributed fashion.

MOBOT-2 also has an onboard arm. Unlike other robots with onboard arms this one has a large workspace enabling it to manipulate objects quite far above the base of the robot. Special care has been taken to ensure that the robot can achieve this reach without tipping itself over.

REFERENCES

[**Brooks 86**] "A Robust Layered Control System for a Mobile Robot", Rodney A. Brooks, *IEEE Journal of Robotics and Automation*, RA-2, No 1., April 1986

[**Crowley 85**] "Navigation for an Intelligent Mobile Robot", James L. Crowley, *IEEE Journal of Robotics and Automation*, RA-1, March 1985, 31–41.

[**Giralt et al 84**] "An Integrated Navigation and Motion Control System for Autonomous Multisensory Mobile Robots", Georges Giralt, Raja Chatila, and Marc Vaisset, *Robotics Research 1, Brady and Paul eds, MIT Press*, 1984, 191–214.

[**Lozano-Pérez et al 83**] "Automatic Synthesis of Fine-Motion Strategies for Robots", Tomás Lozano-Pérez, Mathew T. Mason, and Russell H. Taylor, *International Journal of Robotics Research*, Volume 3, Issue 1, 1983

[**Moravec 83**] "The Stanford Cart and the CMU Rover", Hans P. Moravec, *Proceedings of the IEEE*, 71, July 1983, 872–884.

[**Nilsson 84**] "Shakey the Robot", Nils J. Nilsson, *SRI AI Center Technical Note 323*, April 1984.