Behavior Control for Robotic Exploration of Planetary Surfaces

Erann Gat, Rajiv Desai, Robert Ivlev, John Loch, and David P. Miller

Abstract—**This paper describes a series of robots developed at JPL to demonstrate the feasibility of using a behavior-control approach to control small robots on planetary surfaces. The round-trip light-time delay makes direct teleoperation of a mobile robot on a planetary surface impossible. Planetary rovers must therefore possess a certain degree of autonomy. However, small robots can only support small computers (due mostly to power, not size constraints). Behavior Control provides a means of autonomous control that requires very little computation. The robots described in this paper all used 8-bit, 1-MIP microprocessors with as little as 4 k and no more than 40 k of memory, and extremely simple sensors. Despite these limitations they reliably perform both autonomous navigation and manipulation in both indoor and outdoor rough-terrain environments.**

I. Introduction

RECENT ADVANCES in the miniaturization of scientific instruments [2], [13] have opened the possibility of using very small robots to explore the planets [5]. Because of large round-trip light-time delays (up to 40 minutes for Mars), direct teleoperation of unmanned robots from Earth is not feasible. Planetary rovers must therefore possess a certain degree of autonomy. However, small rovers can only support small computers. In order to capitalize on miniaturized instruments it is therefore necessary to design a control architecture that can adequately control the robot without overtaxing the available computational resources.

The limitation on computation on a mobile robot is imposed primarily by the size of the available on-board power source and not by the physical size of the computer. While it is possible today to build a complete, state-of-the-art 32-bit computer on a handful of chips, such computers require fairly large power supplies. Even the relatively modest demands of low-power CMOS computers can tax the capabilities of space-qualifiable power supplies. Furthermore, the production of space-qualified versions of electronic components is typically many years behind the terrestrial state of the art.

The energy cost of computation is a fundamental problem in the design of small robots for planetary exploration. Energy source technologies are fairly well established, and barring an unforeseen breakthrough, are likely to remain so for some time. There are basically three possibilities. Radioisotope Thermal Generators, or RTG's, produce electricity using thermocouples that are heated by the decay of a radioisotope, usually plutonium. Photovoltaic cells are lighter and less expensive, but usually must be augmented with fairly heavy rechargeable batteries in order to produce acceptable performance. The third possibility is to use non-rechargeable batteries. This provides the greatest power density, and also the shortest lifetime.

Because energy source technologies are fixed, there are only two approaches to minimizing the impact of the demand placed on the power source by the computer. The first is to develop fabrication methods that can produce more efficient CMOS gates. This approach is expensive and time-consuming, and extending such work to produce space-qualifiable parts can be prohibitive. The second is to develop software control methodologies that can operate with smaller, slower computers. Because the power required by a CMOS processor is roughly proportional to clock speed, a decrease in the required computations translates directly into a decrease in the required power, and therefore in the mass and cost of the overall system. This is the option we have pursued.

In the course of our research we have constructed and tested three separate microrovers, ranging from a simple indoor proof-of-concept testbed to a highly sophisticated outdoor testbed. This outdoor testbed carried working science instruments, and is currently the model for a planned Mars mission scheduled for launch in 1996. Subsequent sections describe each of these robots in turn. We begin with an overview of behavior control.

II. Behavior Control

A. Overview

In 1985, Brooks introduced the subsumption architecture, the first and best-known example of what has come to be known as the behavior-control paradigm [4]. Behavior Control is based on decomposing the problem of autonomous control by task rather than by function. Traditional functional decomposition attempts to construct general-purpose functional modules such as world modelers and planners, and advocates connecting them in a serial fashion (e.g., [21], [19]). Behavior control advocates the construction of special-purpose task-achieving modules that are connected directly to sensors and actuators and operate in parallel. These modules are usually called *behaviors*. In order to distinguish between the technical meaning of the word referring to a task-achieving module in

Manuscript received October 27, 1992; revised August 2, 1993. This work was performed at the Jet Propulsion Laboratory, California Institute of Technology under a contract with the National Aeronautics and Space Administration.

E. Gat, R. Desai, R. Ivlev, and J. Loch are with the California Institute of Technology Jet Propulsion Laboratory, Pasadena, CA 91109 USA.

D. P. Miller was with the California Institute of Technology Jet Propulsion Laboratory, Pasadena, CA 91109 USA. He is now with the MITRE Corporation, McLean, VA USA.

IEEE Log Number 9400770.

a behavior-control architecture, and the more generic meaning referring to the physical actions produced by such a module on a real robot, we shall capitalize the former. Thus, a set of Behaviors produces an emergent behavior when running on a real robot. One set of Behaviors might run on two different robots, or on one robot in two different circumstances, and produce two different behaviors.

A number of approaches to behavior control have emerged since Brooks introduced the approach (e.g., [1], [18]). There is considerable disagreement over exactly what characteristics define the behavior control approach, but most implementations have the following features in common. Behaviors (with a capital B) tend to be relatively simple. Complex tasks are achieved by composing many simple Behaviors to produce an emergent complex behavior (with a small b). The composition is accomplished by some form of arbitration mechanism that resolves conflicts when multiple Behaviors attempt to control the same actuator simultaneously. Much effort has been devoted to the design of an arbitration mechanism that will allow component behaviors that have been developed independently to be seamlessly integrated, but no completely satisfactory answer has emerged. Later in the paper we propose a solution based on a fundamental change to the arbitration approach.

There are two significant advantages to behavior control over traditional functional decomposition. First, because Behaviors operate in parallel rather than serially, fast Behaviors need not be delayed by slower ones, allowing the robot to respond to contingencies in real time. Second, because Behaviors are task-specific rather than general-purpose, Behavior designers can take advantage of the structure of the task in order to simplify the Behavior. This is the underlying reason why Behaviors tend to be simple.

The power of this feature of behavior control was dramatically illustrated by one of the early successes of the approach, the problem of collision-free navigation. Brooks demonstrated that collision-free navigation, previously thought to be very difficult and computationally intensive, could actually be achieved with very simple computational mechanisms. In fact, Brooks demonstrated that many complex tasks could be accomplished *reactively*, that is, by simply coupling sensors to actuators through a simple transfer function using little or no internal state (i.e., memory). Connell eventually pushed this hard-core reactive approach to a dramatic limit by building a robot that collected soda cans using no persistent internal state whatsoever [8].

Reactivity has been the subject of much debate and confusion. There are two commonly accepted meanings for the term. The first definition refers to the speed at which a system responds to input. A system is said to be reactive if its response time is small. The second definition refers to the use of internal state. Under this definition a system is said to be reactive to the extent that it minimizes the use of internal state. (Systems that possess both types of reactivity are sometimes called *reflexive*.) Note that the first definition refers to a dynamic property of a system while the second refers to a static property. To distinguish between the two we shall capitalize the latter. Thus, static Reactivity (i.e., minimal internal state) tends to result in dynamic reactivity (i.e., fast response time), but dynamic reactivity does not require static Reactivity, nor does static Reactivity necessarily result in dynamic reactivity. Under either definition, reactivity is not a binary concept but a continuum. Dynamic reactivity is self-evidently a desirable property of a robot control system, but there is a growing consensus that static Reactivity may not be the best way to achieve it under all circumstances.

It should be noted that the issue of reactivity (of either sort) is largely orthogonal to the issue of task-wise decomposition that forms the basis of behavior control. It is often supposed that the minimization of internal state is a *tenet* of the behavior control approach; it is not. Rather, that some behaviors can be implemented this way is one of the *results* of work in behavior control (e.g., [1], [3]). Behaviors need not be Reactive, but they can be and usually are. It has been claimed that traditional AI techniques cannot be used to produce reactive behavior [6], but recent research (e.g., [14], [11] has challenged this claim.

In order to investigate the strengths and limitations of static Reactivity we designed a software infrastructure that would enforce this discipline upon programmers. The centerpiece of this infrastructure is a programming language called ALFA [9]. ALFA is based on the subsumption architecture, but with several significant differences described in the next section.

B. ALFA

ALFA (A Language For Action) is a programming language designed to describe reactive behavior-control mechanisms for autonomous robots. An ALFA program consists of a network of computational *modules* connected by communications *channels*. Channels are themselves computational entities that perform command mediation (cf. [12]).

A module is a computational entity that computes a transfer function from a set of inputs to a set of outputs. Modules are designed to be primarily dataflow devices, but can also support state-machine computations. Every input to a module comes from the output of a channel, and every output from a module goes to the input of a channel. A module makes a connection to a channel by simply referring to the channel by name. Thus, all of the interconnection information is contained within the module definition itself. This allows modules to be inserted and removed at will without the need to reconfigure the communications network.

A channel is a computational entity that accepts an arbitrary number of inputs and combines them into a single output. Unlike modules, channels are purely dataflow entities. The computations that a channel may perform is restricted to operations that are commutative and associative. This restriction guarantees that the output of a channel is independent of the order in which inputs are processed. This restriction still allows a large class of useful combination methods.

Channels may also receive inputs from external sensors, and send their outputs to the outside world to drive actuators. (When ALFA is compiled onto a traditional processor, sensors and actuators are treated as memory-mapped devices.) Thus, the interface between modules and with external hardware is completely homogeneous, allowing modular program development and the construction of abstraction barriers.

The syntax of ALFA is LISP-like. Channels and modules are defined using the following top-level forms:

```
(DEFINE-CHANNEL name input-spec
            { :actuator interface })
(DEFINE-MODULE name . method)
```

A channel's input-spec specifies the source of the channel's input, and how to combine multiple inputs. (The reader is referred to [9] for a more detailed description of the input-spec argument.) The output of a channel may be made available external to the ALFA program by specifying the optional :ACTUATOR keyword. The interface argument is a device-dependent interface specification, usually a label.

A module's method describes the transfer function that it computes. We follow the Scheme convention for describing syntax, in which a final dotted pair denotes an arbitrary number of arguments, in this case a list of commands. The fundamental command used to construct a module's method is the DRIVE command, which sends a value to a channel. Its syntax is straightforward:

```
(DRIVE channel-name expression)
```

The values of channel outputs are accessed by simply referring to the name of the channel as if it were a variable. For example, the command:

```
(DRIVE channel1 (* 2 channel2))
```

sends as one of the multiple inputs to CHANNEL1 a value that is twice the output of CHANNEL2.

The DRIVE command is a dataflow construct. It can be thought of as representing a connection between two nodes in an electronic circuit. In the case of the above example, the connection is made through an amplifier with a gain of 2.

In theory, all of the commands in a module's method run in parallel, and all of the modules in the system likewise operate in parallel. In practice, when an ALFA program runs on a serial machine, the methods of all modules are called in turn. Then all of the channels are updated with the values that result from the new module outputs. The module methods are then called again, and the cycle repeats continuously.

ALFA supports the usual roster of mathematical operations, including arrays and floating point math. In addition, the language supports the LISP LET special form, which allows the creation of lexically scoped temporary variables. However, the semantics of LET in ALFA differ slightly from those of LISP because of the dataflow nature of the language. LET is a syntactic construct that allows the output of a computation to be named so that its value can be referenced more than once. In practice, the effect is the same as a traditional temporary variable, except that its value cannot be changed from the value given to it in the LET expression.

Internal state variables are declared and given initial values using the SLET (State-LET) form. Unlike LET, which simply names the output of a computation, SLET actually defines a storage location whose value persists until it is changed by the program. The values of state variables are modified using the SET command. The syntax of SLET is the same as LET. The semantics differ in that the values given for state variables are initial values only. SLET and SET are the only facilities provided in the language for managing internal state. This repertoire is deliberately impoverished, as ALFA was designed to investigate robotic control algorithms with minimal internal state.

ALFA also supports syntactic abstraction and type abstraction, as well as a number of real-time constructs. For more details, the reader is referred to [9].

An example of a fragment of ALFA code is shown at the foot of this page. This is the actual code used on one of our robots (Rocky III, described in Section IV) to avoid obstacles.

C. Discussion

ALFA is superficially similar to the subsumption architecture, and Brooks' subsequent Behavior Language [5] (as well as Kaelbling's REX language), but there are a number of substantial differences. First, subsumption behaviors are

```
(define-module avoid-obstacles
  (slet ( (state 0) (delay :timer) (direction 0) )
    (if (or left-obstacle right-obstacle)
      (block
        (set direction 4)
        (set state 1)
        (trigger delay 10)))
    (if right-obstacle (set direction - 4))
    (case state
      (1 (drive speed (- fast))
        (drive steer 0)
        (if (not delay)
          (block
            (set state 2)
            (trigger delay 5))))
      (2 (drive speed fast)
        (drive steer direction)
        (if (not delay)
          (set state 0))))))
```

fundamentally state machines. Connell uses these state machines as a substrate to emulate dataflow computations, but the discrete nature of the substrate is not hidden, resulting in cumbersome code.

Second, interconnections among subsumption behaviors are made by means of *wires*, which are a software construct introduced by Brooks. Connell's subsequent implementation is notable in that it used a physically distributed network of processors connected together by physical wires. Wires are somewhat like ALFA channels but with three major differences: 1) a wire has only a single input, 2) messages in one wire can suppress messages in another wire, and 3) the connections between wires and modules are specified in the definition of the wire rather than the definition of the module. This makes sense when one is dealing with physical wires, but as a software construct it leads to a number of problems.

Arguably the most serious problem with wires is that it is never possible to know whether previously developed computations will still work properly when new modules are added, since one can never know when some critical signal will be suppressed [18]. In ALFA, new modules cannot interfere with the internal computations of existing modules. Furthermore, since priorities (when they are used) are set within the definition of a channel, it is possible to guarantee that certain critical pathways cannot be suppressed unless the channel is redefined. In other words, it is not possible to interfere with a working computation in ALFA simply by adding code.

Third, and perhaps most important, ALFA was designed to support a development methodology that is different from the subsumption methodology. Subsumption programs are developed in layers. Higher-level layers interface with lower level ones by *suppressing* the results of the lower-level computations and superseding their results. ALFA was designed to support a layered architecture where higher-level layers interface with lower-level ones by *providing input or advice* to the lower-level layers (cf. [17]). In other words, layers in ALFA provide layers of computational abstraction rather than layers of functionality. This principle will be illustrated in the next section.

III. Tooth

A. Overview

Our first behavior-controlled robot was named Tooth. (See Fig. 1.) Tooth is approximately 30 cm long and 20 cm wide. It has front-wheel steering, two independent rear-wheel drive motors, and a collection of contact sensors and infrared proximity sensors that are used for obstacle avoidance. It also has a two-degree-of-freedom gripper with which it can pick up small objects. The robot has five actuators: two drive motors, the steering motor, and two motors to control the gripper, one to open and close it, and the other to raise and lower it.

There are ten one-bit sensors: five collision contact sensors (one at the tip of each gripper arm, one in the center rear of the gripper, and two connected to a rear bumper), a pair of gripper contact sensors (one inside each of the two gripper arms), two forward-looking infrared proximity sensors, and one infrared break-beam sensor for detecting when an object is in the gripper. There are also ten analog sensors: eight cadmium-sulfide photo cells (CdS cells) (used to sense the location of a light-bulb beacon) and a tachometer on each drive motor.

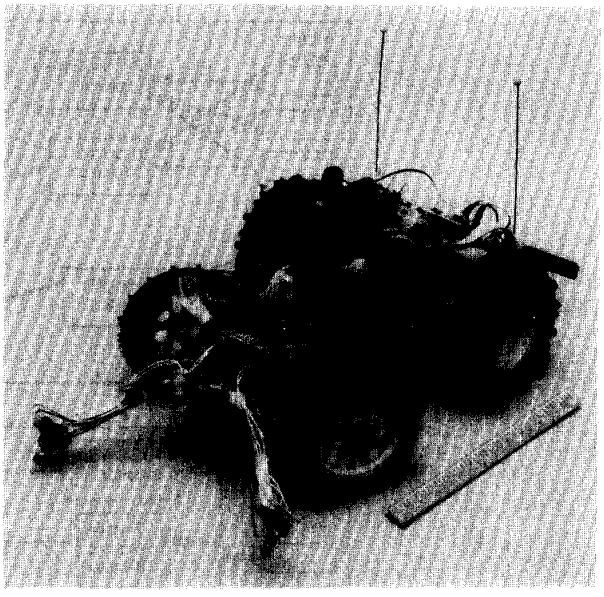


Fig. 1. The Tooth microrover.

The robot's computing resources consist of two Motorola 6811 microcontrollers, each with 2 k bytes of electrically erasable programmable read-only memory (EEPROM) and 256 bytes of random-access read-write memory (RAM). Only about 3.5 k of EEPROM and 100 bytes of RAM were actually used (and most of the used RAM was memory allocated by the compiler for temporary storage).

Due to its mechanical design, Tooth is exclusively an indoor robot. It was used to demonstrate a proof-of-concept "coffee-cup sample-return mission." The robot's objective was to explore the perimeter of an office, pick up any small objects it found there, and deposit them next to a beacon (a 150-watt light bulb).

B. Control Structure

The ALFA program for guiding Tooth was distributed over Tooth's two processors. (See Fig. 2.) The first processor was connected to the CdS photocells and the controls for the gripper, and the second was connected to the drive and steer motors, as well as the bump sensors and infrared proximity sensors. These processors are therefore referred to as the grasp processor and the drive processor respectively. Communications between the two processors takes place over a serial link. This fact is completely transparent to the ALFA program. The serial link simply transmits data from a channel on one processor to a corresponding channel on the other.

The control system for Tooth follows a bottom-up design methodology [11]. Control is organized into layers according

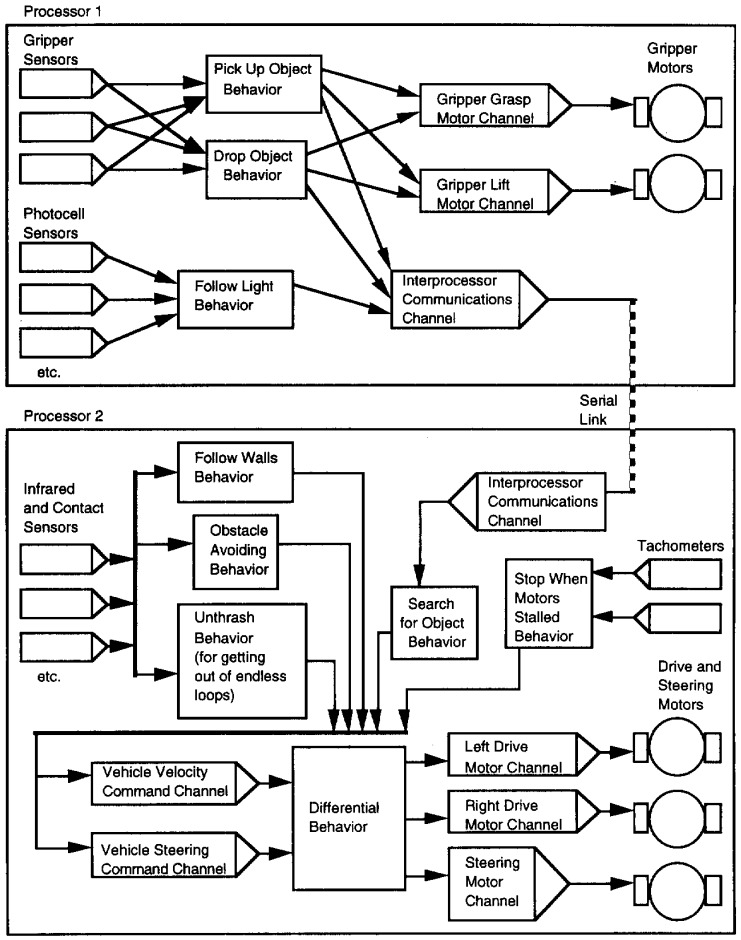


Fig. 2. The structure of Tooth's control program.

to level of abstraction. Each layer interfaces to the layer above it by means of channels that play the role of virtual actuators to the upper layer and virtual sensors to the lower one. As one goes up in the hierarchy, the virtual actuators control the robot at higher levels of abstraction.

The drive processor contains the first two layers of the control hierarchy. The first layer consists of a single module for coordinating the drive and steering motors to provide a "software differential." The interface to this layer consists of two virtual actuators that control the vehicle velocity and heading. The values of these actuators are used by the differential behavior to drive the actual motors, which also appear to ALFA as channels.

The second layer contains a variety of modules for dealing with various contingencies of getting around in the world. These include veering away from or backing up from obstacles, detecting and recovering from stalled motors, getting out of endless loops, and getting out of dead ends (see Section III-C). The interface to this layer consists of a single channel that indicates a preferred direction of travel.

The third layer, implemented on the grasp processor, has modules for picking up an object, dropping the object when near the beacon, and directing the vehicle towards or away from the beacon.

The robot was able to tell when an object was the proper size and weight for it to lift by monitoring the two side gripper sensors. Objects that are too small to grasp will not activate these contact sensors. There is also a contact sensor at the rear of the gripper that we call the choke sensor. This sensor is used in conjunction with the motor tachometers to tell when the robot is attempting to move an object that is too heavy.

One of the consequences of our architecture and development methodology is that it is possible to make certain guarantees about the robot's behavior. For example, the second layer in Tooth's control system prevents the robot from moving forward when obstacles are detected by both infrared proximity sensors. This behavior is guaranteed regardless of the structure of any higher layer. In other words, the robot's obstacle avoidance behavior is guaranteed not to change regardless of what changes are made to the third control layer. Another example is that the robot's drive motors are guaranteed to have the proper differential velocities around turns regardless of what changes are made in the second or third control layers. These guarantees are only possible

because of the restricted interactions between layers. In the subsumption architecture it is usually not possible to make guarantees about the behavior of the robot when a new layer is added or an existing layer is modified, since the new layer may suppress any communications wire in the layer below.

C. Getting Out of Loops and Dead Ends

One of the problems with purely Reactive control is that it can get the robot stuck in loops. Although random perturbations will eventually get a Reactive robot out of many loops, the addition of a small amount of internal state can dramatically improve performance. There are three modules in Tooth's code that illustrate this. The first is the "unthrash" module in the second layer. This module counts the number of times the robot changes direction in a certain period of time. If this count exceeds a threshold, the module steers the robot in a random direction. This module intentionally introduces a random perturbation in cases where the robot is stuck in a short-duration loop. The unthrash module has a lower priority than the obstacle avoidance module, and so does not introduce a risk of collisions.

The second example is the obstacle-avoidance module, which includes a code fragment that allows the robot to escape dead-ends. Whenever the robot is forced to back up (which occurs when both IR proximity sensors are activated simultaneously, or the forward collision sensors are activated) the amount the robot backs up is recorded. If, once the robot has resumed forward motion, the robot is forced to back up a second time before it has moved forward longer than it has moved backwards, then the amount of time it moves backwards is increased. Eventually, the robot backs up far enough to extricate itself from the dead-end.

The third module that employs internal state to improve performance is the module that grasps an object. This module keeps count of the number of attempts the robot makes to grasp an object. The robot has contact sensors on the inside of the gripper that indicate when the robot has a good grip on an object. After five unsuccessful attempts to grasp the same object, the robot backs away and begins to search for a new object.

There is also a fourth technique used on Tooth to help escape endless loops that does not involve the use of internal state. The turning radius used when the vehicle is moving forward are slightly different from that used when backing up. In situations where the robot is moving back and forth between two closely spaced obstacles this technique causes the robot's orientation to precess, and eventually allows the robot to escape the cycle.

D. Results

Fig. 3 shows the layout of a typical test course and the beginning of one of our experiments. The figure was constructed by hand from a video tape. The shaded areas are obstacles and/or walled off areas (e.g., a desk, a trash can, etc). The white circle is the beacon. The small black circles are the objects to be retrieved. We used styrofoam coffee cups on occasion, but most of the time we used small cylinders made of plastic and foam that we dubbed "Tooth toys."

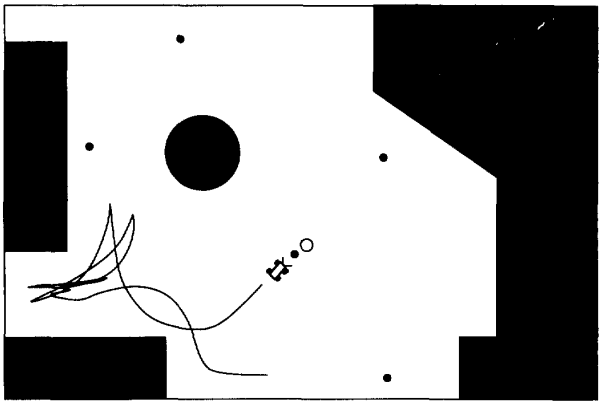


Fig. 3. The test course and the path taken to retrieve the first object.

When searching for toys (i.e., when the robot's gripper is open), one of the robot's Behaviors moves it away from the beacon. This, combined with the obstacle avoidance Behavior, results in an emergent behavior that causes the robot to explore the perimeter of the test course. Because the robot has no way of sensing toys remotely they had to be placed near the perimeter of the course where the robot would naturally come across them in the course of its wanderings.

In the experiment pictured, the robot followed the perimeter of the test course until it encountered a toy near the lower left hand corner. It picked up the toy, extricated itself from the dead-end, and then dropped the toy off near the light. The events shown in the figure took less than one minute. Tooth was able to acquire all the toys and deposit them by the light within eleven minutes with only minor human intervention. We intervened only in cases where Tooth would nudge a toy out of its original position, either because its gripper was already holding a toy, or because the object was contacted by the tip of the gripper or by one of Tooth's wheels. (The robot would nominally avoid toys that were not correctly positioned to be grasped, but the infrared sensors are not infallible, and the toys are not heavy enough to trigger the forward contact sensors.) The intervention consisted only of moving the toys back into their original location near the perimeter of the course.

We experimented with Tooth on this test course and others under a variety of circumstances. Under most conditions the robot's behavior was very robust. It would reliably acquire and return the objects it found to the vicinity of the beacon. We found only two situations that the robot could not deal with: non-somatic[1] obstacles, such as wires or holes, and bright external light sources that would flood the robot's infrared

[1] A non-somatic obstacle is an environmental feature that impedes the robots's movement by virtue of some property other than its physical extent. An ordinary obstacle such as a wall is a somatic obstacle because a robot cannot occupy the same space as the wall. A phone cable is a non-somatic obstacle because a robot usually can move to occupy the same space, with the side-effect of displacing the cable. Nevertheless, the cable is an obstacle since the robot can become entangled. Other examples of non-somatic obstacles are dust-filled holes, slopes of material at the angle of repose and bodies of water. Non-somatic obstacles can be very difficult to detect since they are often geometrically identical to areas that are safe to traverse.

sensors. Both of these problems could be overcome by the addition of appropriate sensors.

There is one additional result from this experiment. It was found after the experiment was complete that the left-facing CdS cell was not working. This explained why Tooth sometimes took slightly longer than expected to turn towards the beacon. It also lends some weight to the claim that behavior control is robust in the face of hardware failure.

This preliminary demonstration illustrates many of the characteristics of the behavior control approach. The robot does not follow paths that are "optimal" in any usual sense of the word. However, its actions are reasonable, and it does get the job done using very little computation and very simple sensing. Tooth is reliable enough that we regularly use it for impromptu demonstrations for visitors, whom we allow to interact freely with the robot. To our knowledge, among robots that perform both autonomous navigation and manipulation, Tooth is by far the most reliable. Although we have not kept an accurate log of all the demonstrations we have performed with the robot, there have been well over a hundred spanning a period of over two years. In all that time we observed only a single failure,[2] which was not due to one of the two factors noted above (non-somatic obstacles or bright ambient light).

Fig. 4. The Rocky III microrover.

IV. Rocky III

In order to produce a more convincing demonstration that behavior control could be used in a realistic planetary mission, we constructed a series of robots to operate outdoors in rough-terrain environments. The first of these robots, Rocky I, was a mobility testbed only, and was never operated autonomously. Rocky II was never constructed. This section describes the next robot in the series, Rocky III (pictured in Fig. 4), which successfully demonstrated a sample-return mission in an outdoor rough-terrain environment [15].

A. Overview

The chassis design of all the Rocky robots is a six-wheeled springless suspension system called the "rocker-bogie," which consists of two pairs of rocker arms or "bogies." Each pair consists of a main rocker arm and a secondary rocker arm whose pivot point is at the front end of the main arm. Adjustable mechanical stops limit the rotation of the secondary arms. The two rocker-arm assemblies are connected through a differential gear (see Fig. 5). The main body of the robot is mounted to the housing of the differential. The pitch of the main body is thus the average pitch of the two rocker-arm assemblies, providing a stable mount for instruments and sensors. An electronics enclosure is mounted on the main body. The vehicle is powered by rechargeable lead-acid batteries that are mounted below the differential. The mass of the batteries and the wheels and motors keeps the center of gravity at the differential or slightly below. There is a full wheel diameter of clearance between the ground and the bottom of the battery housing.

The robot's six wheels are thirteen cm. in diameter, and are independently driven by motors mounted inside the wheels. The four front and rear wheels are independently steered. Ackerman steering is used, theoretically allowing the robot to turn about any point along a line going through the two middle wheels, including the robot's center[3].

At the front of the robot is mounted a three-degree-of-freedom arm. The end effector is a soft-soil scoop. The arm can reach approximately five centimeters below the plane of the wheels in front of the robot, and folds to rest on top of the electronics enclosure when the robot is in motion.

Like Tooth, Rocky III has very simple sensors. A flux gate compass is mounted on a mast above the main body of the robot to keep it away from the electromagnetic interference generated by the motors. The compass element is mounted on a float so that changes in roll and pitch up to twenty degrees do not affect its heading reading. The compass is accurate to approximately one degree of arc. Roll and pitch inclinometers in the main body of the robot are accurate to about half a degree. Magnetic reed switches installed on the front-bogie pivots indicate when one of the rocker arms is at one of its limit positions. The two middle wheels are instrumented with one-count-per-revolution encoders that are used to implement extremely crude dead-reckoning. Finally, there are mechanical contact sensors underneath the robot's bottom panel, and at the front of the robot.

Rocky's arm has eight-bit position encoders on each of the three joints. Inside the end-effector are two contact switches used determine the hardness of the soil. One switch has a spike attached to it, the other has a flat plate. When the gripper

[2] That failure turned out to be caused by a program bug that disabled the escape-from-dead-end behavior under obscure circumstances involving the precise placement of the beacon, an obstacle, and a duct-taped extension cord.

[3] On Rocky III, the steering motors are not strong enough to turn the wheels unless the robot is moving, making it impossible to realize a turn about a point inside the perimeter of the robot. However, Rocky IV, using the same mechanical design, is able to turn about its center (see Section V).

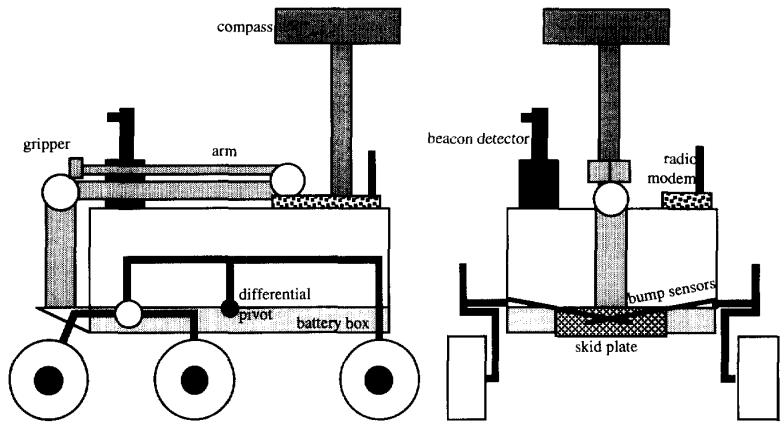


Fig. 5. Side and front views of Rocky III.

is opened and pressed against the ground, the spike makes contact before the plate. If the soil is hard, the switch with the spike will close on contact. If the soil is soft, then the spike penetrates the ground and the switch with the flat plate closes first.

Rocky also has an infrared beacon detector mounted on a rotating platform that senses a homing beacon mounted above the sample receptacle (i.e., the "lander"). The beacon consists of between three and five sets of infrared LEDs operating at distinct frequencies that can be discriminated by the detector. The LED optics are arranged so that each set of LED's is visible only within a certain range of headings relative to the lander. (See Fig. 7.) The beacon detector can also determine the angle to the beacon using an eight-bit encoder on the robot, although this information was not used in the experiments. The beacon has a range of approximately five meters.

The computer on Rocky III was a single 6811 processor equipped with 32 k bytes of memory, though only about 10 k bytes were used. In addition, the system contained four interface boards for controlling the robot's motors and sensors, two boards for power distribution and conditioning, and some miscellaneous electronics. All of the boards were either off-the-shelf commercial products, or constructed in-house using single-sided PC boards or wire-wrapped by hand. The compass and the processor board, both commercial products costing a few hundred dollars each, were by far the most complex electronics on the robot.

B. Control Structure

The structure of the control software for Rocky III is shown in Fig. 6. Note the similarity of this control structure to that of Tooth. The three layers are nearly identical. The bottom layer provides an abstract interface for controlling the robot's speed and direction. In the case of Rocky III, this is considerably more difficult than Tooth. First, there are ten motors to control rather than three (six drive motors and four steering motors), and Ackerman steering requires considerably more computation than is required for a two-wheel differential. However, the appearance of the bottom layer to the rest of

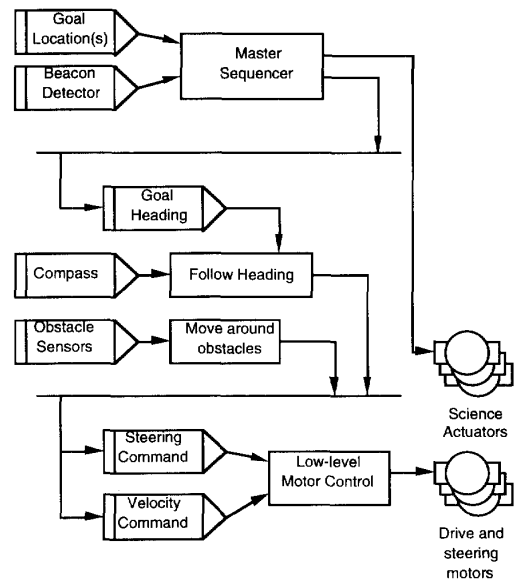


Fig. 6. The structure of the control software on Rocky III.

the system is absolutely identical with that of Tooth's bottom layer, i.e., two channels that appear to directly control the vehicle's speed and direction.

The second layer, as on Tooth, is responsible for obstacle avoidance. Because Rocky III has no look-ahead sensors, it can only detect obstacles if it actually runs into them. Moving around obstacles is accomplished by backing away from the obstacle and turning to one side. This is a fairly simplistic strategy and could be improved, but field tests have shown it to be surprisingly effective.

The third layer is responsible for high-level control over the mission. On Rocky III, this layer acts as a simple sequencer, which moves the robot through a series of way-points (goal locations), stops the vehicle, collects a soil sample, and returns to the lander.

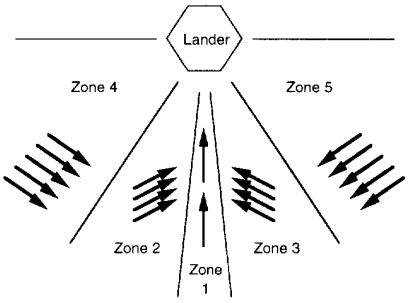


Fig. 7. The docking strategy used on Rocky III.

The robot uses the following strategy to return to the lander. If the robot is not within range of the beacon, it moves towards the lander by dead reckoning. Once the robot detects the beacon it moves along a heading determined by the beacon frequency that the robot detects (i.e., the zone in which the robot finds itself). (See Fig. 7.) The headings are given as offsets from the lander's orientation, which is determined from the compass reading at the start of the mission. It is therefore required that the rover be aligned to the lander initially. In an actual mission this would, of course, happen automatically. If the robot loses sight of the beacon, it continues on its previous heading for a few seconds in an attempt to reaquire the beacon before reverting to dead-reckoning. Arrival at the lander is indicated by the robot detecting beacon Zone 1 and a forward contact sensor being activated simultaneously.

C. Results

The goal of Rocky III was to demonstrate a proof-of-concept prototype of a small rover that could meet the requirements of a simple scientific planetary mission. These requirements include: the ability to navigate to a designated area; the ability to acquire a suitable sample of planetary material; the ability to negotiate obstacles, either by avoiding them or surmounting them; the ability to return to the lander with sufficient precision to deposit the sample there; the ability to operate with no real-time communication; and the ability to carry all power, computation, communications, etc. on board. Rocky III was not intended to provide an exhaustive scientific assessment of the parameters of such robots, and we had neither the time nor the resources to perform such experiments. We therefore have only anecdotal data to report.

Several dozen end-to-end missions were performed to verify the robot's abilities. (As with Tooth, we did not log every experiment and so we do not know the exact count.) The experiments took place in a large indoor laboratory, and outdoors in the Arroyo Seco outside of JPL, a dry wash strewn with rocks, boulders, sand, and hard packed soil. All of the experiments had the same basic format, though the details of the robot's starting position and orientation, the positions of obstacles, the sample site and way-points differed from test to test. In the indoor experiments the sample site was located approximately three meters from the lander. In the outdoor experiments, the sample sites were approximately ten to twenty meters from the lander.

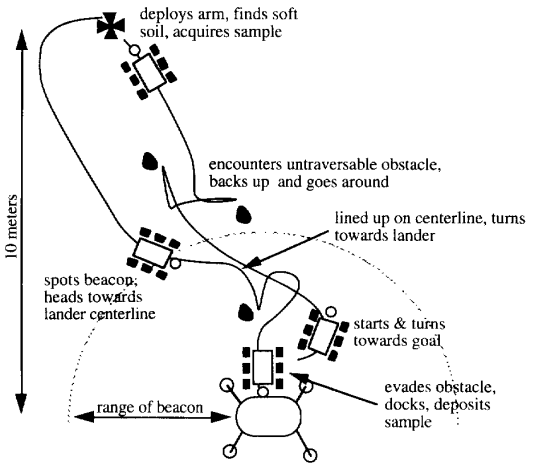


Fig. 8. A typical Rocky III experiment.

Fig. 8 shows the course of a typical experiment. The rover starts by moving away from the lander and turning towards the goal, navigating by dead-reckoning. If it encounters an obstacle that it cannot easily traverse (i.e., one that causes either the pitch or roll sensor to go outside of acceptable limits, or that triggers one of the bogie limit magnetic reed switch sensors) the robot executes an avoidance maneuver. Upon reaching the sample site, the robot deploys the arm and searches for a suitable soil sample. The robot moves forward by small increments until a patch of soft soil is found, at which point the scoop closes and the arm is retracted. The robot then returns to the lander by using the docking strategy described earlier.

The reliability of the robot is remarkably high, successfully completing its mission over 90% of the time. Rocky III is sufficiently reliable that we are comfortable giving "live" demonstrations to visitors, and even invite them to stand in the robot's way. After the software was debugged and tuned, dozens of runs were performed with only a few failures, most of which involved hardware failure. In all cases the avoidance software succeeded in getting the robot through the obstacles and to the destination. Even during a run where the sample area was (inadvertently) designated in a heavily vegetated spot, the robot eventually made its way around the large weeds and to the sample area.

There are a number of areas in which the performance of Rocky III could be improved. Most of the potential improvements are to the hardware, which was built on an extremely tight schedule and budget and is not very reliable. The addition of some sort of remote sensing of obstacles would allow the robot to move much more efficiently.

There are two potential improvements to the current docking strategy. First, the robot uses its front contact sensors to detect when it has reached the lander. Thus, a very large obstacle can be mistaken for the lander under certain circumstances. This could be easily overcome by the addition of an additional sensor to unambiguously identify contact with the lander (e.g., a short-range beacon or a magnetic contact sensor). Second,

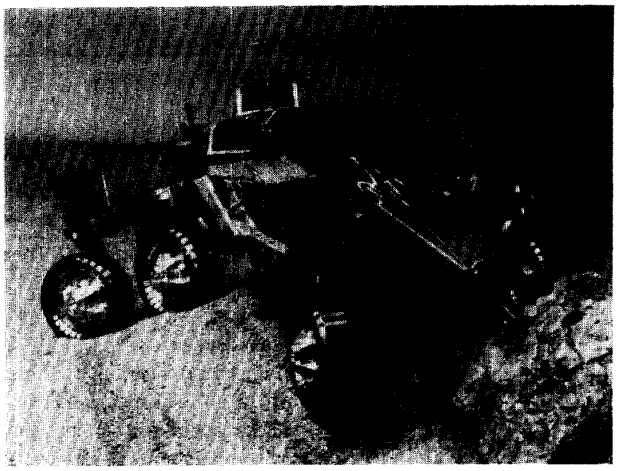


Fig. 9. The Rocky IV microrover.

there are some instabilities in the docking strategy in the region near the lander. These instabilities could almost certainly be eliminated by more careful design and analysis of the system.

Rocky III is the first example known to us of an autonomous robot that operates in outdoor natural terrain and performs both navigation and manipulation. (The only other example is Rocky IV, described in the next section.) The robot's performance has been demonstrated in dozens of tests in both an indoor laboratory setting and outdoor rough-terrain environments, though as of this writing no controlled study of the robot's abilities and limitations has been performed.

V. Rocky IV

The success of Rocky III prompted JPL to conduct a full-scale demonstration of a microrover performing a scientific mission under realistic conditions using functional science instruments. The result of this work was Rocky IV, which was initially demonstrated in June of 1992.

A. Overview

The chassis of Rocky IV is functionally identical to Rocky III. The robot has the same six-wheel springless "rocker-bogie" suspension system, and the footprints of the two robots are the same size. However, there are a number of significant differences. In Rocky IV a differential beam replaces the differential gear used on Rocky III. (See Fig. 9.) The main body of the robot is mounted to the main bogie pivots as well as the differential beam pivot. As on Rocky III, the pitch of the main body is thus the average pitch of the two rocker-arm assemblies. There is a full wheel diameter of ground clearance below the robot's main body. The drive and steering configuration is identical to Rocky III; Ackerman steering is again used. Advanced design and fabrication techniques have dramatically reduced the mass of the robot; whereas Rocky III masses about 18 kg, the mass of Rocky IV is 7.5 kg. The robot's tires are made of spring-steel. Rubber tires would become brittle and shatter in the cold temperatures found on Mars.

The robot's main structural element is an aluminum cylinder that connects the two main bogie pivots. This cylinder divides the main body into forward and rear sections. The rear section contains the robot's computer and battery compartments, while the forward section contains the robot's scientific payload.

The main computer on the robot is a Motorola 6811 microcontroller with 40 k of memory, although only about 20 k was used. Three Intel 8751 microcontrollers are connected to the 6811 via a synchronous serial bus and act as slave processors in order to extend the number of I/O pins available. We believe that this general architecture—a number of slave processors connected to a master processor over a high-speed synchronous serial bus—is a sound design for a small robot, which typically requires more I/O lines than a single processor can provide.[4]

The robot's payload of science microinstruments consists of a visible spectrometer mounted on a tilt mechanism, and a chipping tool used to remove the outer layers from rocks to expose unweathered material. The chipper is also used to deploy a microseismometer, but this instrument was not electrically connected to the robot. The spectrometer assembly also contains a small color video camera. This camera shared an optical path with the spectrometer making it possible to determine unambiguously the source of spectral data.

The following navigation sensors were used on the robot:

1) Four custom-made infrared proximity sensors mounted on the steering motors, one at each corner of the robot.
2) A contact sensor mounted along the front edge of the main body.
3) Pitch and roll inclinometers.
4) Joint encoders (actually potentiometers) mounted at the main bogie pivots and the differential pivot.

[4]Though the basic design was sound, there are incompatiblities between the Intel and Motorola implementations of the synchonous serial bus that caused substantial difficulties. In particular, the 8751 cannot operate as a synchonous slave (i.e., it cannot receive the synchronous serial clock as an input), a problem that required substantial software development effort to circumvent.

5) A magnetic flux-gate compass mounted on top of the computer housing. This $500 off-the-shelf unit (made by KVH inc.) was by far the most complex sensor used on the robot.
6) Incremental single-phase encoders on all six drive wheels.
7) A current-threshold sensor on the chipper's main drive motor, used to detect stalls.
8) An infrared beacon detector.

The following paragraphs describe each of these sensors in turn.

The infrared proximity sensors are active sensors consisting of an infrared LED modulated at 40 kHz and an integrated detector/demodulator unit. The detector is mounted in a plastic housing approximately 1.5 cm × 1 cm × 3.5 cm. One of these sensors is mounted at each of the robot's four corners. The sensors are mounted with Velcro, making it quite easy to adjust their position, but nearly impossible to align them accurately, repeatably, or reliably.

The infrared proximity sensors are used to detect rocks and other large obstacles so that the robot can avoid contacting them. The range of the sensors depends on a number of factors, including the size and albedo of an obstacle, and the amount of background light. In direct sunshine the sensors can reliably detect light-colored obstacles at a distance of about 25 cm. Indoors the range more than doubles. Dark objects, however, are all but invisible to these sensors, especially outdoors.

The infrared proximity sensors provide a first line of defense against collisions with obstacles, but they are insufficient to allow the robot to avoid all obstacles. Large areas of the robot's perimeter are not covered by the IR sensors, most notably the area directly in front of the robot. This area is particularly difficult to cover because of the need not to obscure the view of the scientific instruments. A partial solution to this problem is a contact sensor mounted on the front of the robot's main body. This sensor consists of a flexible metal plate attached to the robot along one edge. Along the other edge is a set of small switches that are activated when there is pressure along that edge of the plate. This sensor serves to detect actual collisions with obstacles along the front edge of the robot.

The pitch and roll inclinometers, along with the joint encoders, are used to detect hazards such as holes and steep slopes that can be detected by neither the IR sensors nor the contact sensor. The information from the inclinometers and joint encoders can theoretically be used to determine the exact position in space of all six of the robot's wheels. However, this requires a significant number of floating point computations that represent a considerable demand on the robot's processor. Instead, the robot navigated by setting hard limits on the values of each of these sensors individually and treating any violation of these limits as an obstacle. It turned out that this made the robot overly conservative, as there are large classes of situations where individual limits are met but that do not represent a hazard.

For example, consider the situation depicted in Fig. 10. It would seem that a check on the position of the secondary bogie arms should allow the detection of cliffs. (See Figs. 10(a) and 10(b).) However, scaling a small obstacle can cause the secondary arm to reach the same position as it does at a cliff (Fig. 10(c)), resulting in the embarrassing behavior of backing off of an obstacle after it has been successfully surmounted. These situations can be distinguished, but only by looking at combinations of sensors. Deciding which combinations of sensors to look at is an active area of current research. Another possible method for detecting cliffs reliably is the addition of strain gages to monitor the normal force on the wheels, or some form of look-ahead sensor.

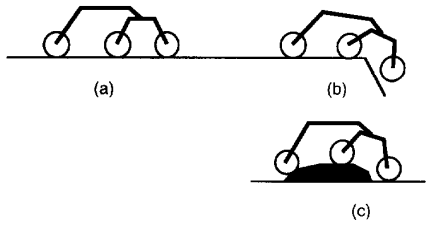


Fig. 10. A cliff (b) cannot be distinguished from a moderate obstacle (c) by articulation encoders alone.

The flux-gate compass and the wheel encoders were used to implement a dead-reckoning system to track the robot's position. Only the middle wheel encoders were used. On level ground where the robot's wheels did not slip much, the dead-reckoning accuracy was measured at better than 5% of distance traveled. However, on slopes or in very loose soil, the dead-reckoning error is potentially much larger. One possible method for correcting for accumulated dead-reckoning error is to use a pair of stereo cameras mounted on the lander in order to locate the robot. Such a system was used to designate destination points during our experiments, but it turned out to have unacceptably large errors due to calibration difficulties.

It should be noted that one of the advantages of the behavior-control approach is that it allows the rover to perform useful tasks even in the face of extremely large dead-reckoning errors.

The navigation strategy used is the following. When no obstacles are detected the robot turns straight towards a designated goal. If a single forward infrared sensor is activated, the robot veers away from that side. If both forward infrared sensors are activated, the robot turns in place. If the contact sensor is activated, or any of the other sensors exceed their preset limits, then the robot backs up a short distance, turns approximately 45 degrees, and then goes forward a short distance. This basic strategy was demonstrated to be quite effective on Rocky III, but because of the touchy nature of the articulation sensors, Rocky IV would often back away from phantom obstacles. This is primarily a shortcoming of the sensor processing strategy, and not of the sensors themselves, nor of the navigation strategy, although these could also be improved.

The chipper stall sensor was used to implement automated chipping. In order to be effective, the chipper must be placed at a particular distance from a rock. Rather than try to push the limits of the robot's positional accuracy, the following strategy was used. The robot was first positioned in front of a rock. The chipper was then activated, and the robot drove slowly

forward until the chipper stalled. The robot then drove slowly backwards. In order to be effective, this "chipper dance" had to be performed repeatedly in order to keep the chipper in contact with the rock without getting stuck on protrusions. As long as the robot moved slowly the strategy worked quite well.

The beacon detector was intended to be used to allow the robot to return to the lander to deposit soil samples. However, because of time constraints, docking was not implemented and this sensor was never actually used. The beacon system is essentially identical to that used on Rocky III.

B. Status

The software currently on the robot is preliminary, and consists mostly of code taken directly from Rocky III. As of this writing, the beacon-following behavior has not yet been implemented, and we have not had the opportunity to collect any controlled data on the robot's performance. Nevertheless, Rocky IV has been demonstrated in a simulated mission scenario in a rough-terrain outdoor environment. In this scenario, the rover disembarked from a mock-up lander, deployed the micro-seismometer, navigated autonomously to a rock designated by human operators, chipped the weathering rind off the rock, and collected a soil sample. This scenario and the environment are very close to an actual Mars mission. In fact, rocks were imported to the site to make the rock distribution similar to that found at the Viking lander sites on Mars.

In addition to the official demonstration, this sequence was performed several dozen times as practice runs. None of the runs were flawless. However, every segment of the scenario has been successfully demonstrated multiple times during one run or another. The vast majority of the failures were caused by unreliable hardware. (For example, activating the rock chipper often causes the computer to crash.) Nevertheless, the autonomous navigation software appears to be fairly reliable, though anomalous behavior is sometimes observed. Some of these anomalies can be attributed to flaky hardware (e.g., setscrews slipping on the bogie joint encoders). Others are caused by magnetic anomalies that affect the compass, causing the robot to take circuitous paths to a goal. This would not be an issue in a real mission. Since Mars has no magnetic field, a laser gyro would be used instead. Work is currently underway to explain and correct the remaining anomalies. The computer system is currently being replaced by the computer to be used on an actual mission, which is based on an Intel 8085 processor.

VI. Discussion

In recent years the behavior control approach has been demonstrated in a wide variety of circumstances. But one question that has never been satisfactorily answered is why behavior control appears to work as well as it does. To answer this question in a scientific manner is much more difficult than is generally appreciated. This is because behavior control is, ultimately, a design methodology, and evaluating design methodologies is an endeavor that has always been fraught with difficulty. It is very hard to separate the effects of methodology from those caused by programmer skill, algorithmic issues, or any of the myriad variations that can occur in the experimental setup, especially in an outdoor environment.

We can, however, make some informal observations. In general, our experiences support David Chapman's conclusion that behavior control succeeds largely because action selection is simply not a very difficult problem [7]. Chapman writes:

> [O]ur first papers were about [the action selection] problem; we came up with ideas like subsumption, goals as parallel program specifications, dependency networks, and action arbitration.
> Which of these alternatives is better? This turns out to be a non-question. They all work fine; none of them is very hard to use because choosing what to do just isn't difficult. It seems that any scheme that is reasonably programmable will work fine.

The sheer number of successful *ad hoc* solutions to the navigation problem is evidence to support Chapman's claim. The problem space is rich with working solutions. Behavior control is a methodology that allows a designer to locate one of these solutions quickly. This is largely due to the short development cycle, which allows the designer to quickly test many design iterations.

ALFA facilitates the exploration of the design space through its homogeneous communications and abstraction barrier facilities. Our experience has been that ALFA code is very easy to write and debug, and is often reusable even across robots. All of the ALFA code for Rocky IV, for example, was written by one programmer in about two weeks, and reuses much of the code used on Rocky III. Most of the programming effort in Rocky IV went into writing the device drivers.

ALFA also imposes to a certain extent the discipline of limiting the use of internal state. We have found this "feature" of the language to be a mixed blessing. On the one hand, we rarely encountered internal-state-related bugs. On the other hand, in circumstances where internal state was necessary (e.g., when doing mission sequencing) the resulting code was often very cumbersome. We are currently working on a successor to ALFA that provides more facilities for managing internal state while retaining ALFA's compactness and ease of use.

It is tempting to draw some conclusions about the relative size and complexity of the robots described in this paper compared to that of planetary rovers based on more traditional control methodologies such as JPL's Robby or CMU's Ambler, which are many orders of magnitude larger, heavier, more complex and more expensive. One must be very cautious about one's conclusions. It is true that traditional sense-plan-act methodologies require more computational resources, which ultimately result in larger, more complex and expensive robots. However, not all of the differences between the Rocky robots and Robby and Ambler can be attributed to the control methodology. Both Robby and Ambler were prototypes for the Mars Rover Sample Return mission for which one of the mission requirements was that the vehicle be able to scale 1-meter-high obstacles. At least part of the increased size of these

two robots can be attributed to this mechanical requirement. It is not known how large these robots might have been had this requirement been relaxed, though it seems clear that they could be considerably smaller than they are. However, it seems equally clear that it is not possible to build a robot controlled by a standard sense-plan-act control architecture that is as small and light as Rocky IV, especially if one is constrained to space-qualified hardware.

It is equally tempting to draw some general conclusions about the complexity of the sensors required to support a behavior control architecture. Brooks argues for large numbers of sophisticated sensors, but our robots seem to do quite well using only a very few simple sensors. We have demonstrated that complex sensors are not necessary for one particular application, but to draw broader conclusions would require more data. As we set our sight on more complex tasks the sophistication of both sensors and control strategies will almost certainly have to increase, though we conjecture that the rate of this increase could be surprisingly low.

VII. Summary and Conclusion

We have described a series of three robots constructed at the Jet Propulsion Laboratory to demonstrate the utility of small mobile robots using behavior control for planetary research and exploration. One of the primary technical challenges is to design a computational infrastructure that can operate successfully with the extremely limited power available aboard a small rover. There are two mutually complimentary ways to address this problem. One is to develop low-power computational hardware, and the other is to develop control methodologies with low computational demands. In this work we have focused on the latter approach.

We have investigated the behavior-control paradigm as an alternative to traditional sense-plan-act control methodologies. Behavior control is based on the principle of taskwise decomposition, in contrast to the traditional functional decomposition. Taskwise decomposition permits simplified control software because individual modules are task-specific and not general-purpose. Task constraints can therefore be used to simplify their design.

We use a modified version of Brooks' subsumption architecture, embodied in a programming language called ALFA. The primary difference between our architecture and Brooks' is in the manner in which layers interact. In subsumption, higher level layers suppress communications in lower-level layers. In our architecture, higher-level layers provide information or advice to lower-level layers. ALFA is also different from subsumption in that it separates dataflow computations from state-machine computations, and in the way in which computations are connected to each other. ALFA uses a computational construct called a channel to do command mediation among modules. Both computations and interconnections are specified within module definitions, allowing modules to be inserted and removed without restructuring the communications network. Although none of the robots described in this paper use maps or other higher-level computations, the architecture is designed to support such augmentations, and we have conducted some experiments that integrate maps and strategic planning into the architecture [11] (cf., for example, [17]).

We have constructed three small behavior-control robots: Tooth, an indoor robot, and Rocky III and IV, which are outdoor testbeds. Rocky IV was equipped with functional scientific micro-instruments.

Tooth demonstrates that it is possible to construct a reliable behavior-controlled micro-rover that autonomously performs a complex task involving both navigation and manipulation. Rocky III demonstrates that it is possible to construct a reliable behavior-controlled micro-rover that performs a complex task involving both navigation and manipulation under realistic mission-like conditions in rough outdoor terrain.

The results of Rocky IV are still preliminary, but initial indications are that Rocky IV will extend these results to a robot with functioning scientific instruments. Rocky IV is currently the baseline design for a Mars microrover that is scheduled to be launched as part of the Mars Environmental Survey (MESUR/Pathfinder) mission in 1996.

Collectively, these robots demonstrate that neither strategic planning nor complex, carefully calibrated sensors are necessary to produce robust intelligent behavior in autonomous robots. The tasks performed by these robots are as complex (or more) than the tasks performed by state-of-the-art robots controlled by more traditional methods. Furthermore, Tooth and Rocky III are extremely reliable.

However, just because planning is not necessary does not imply that it is undesirable. In other work we and other researchers have described various ways in which traditional planning-based approaches can be combined with behavior control in order to improve overall performance (e.g., [16], [20], [11], [19]).

We have also used ALFA and our advice-based control architecture on two other robots not covered in this paper. Details may be found in [10] and [11].

Acknowledgment

The work described in this paper was the result of a large team effort over the course of nearly four years. Colin Angle built Tooth and wrote the initial software for it using Rod Brooks' subsumption compiler. Brian Yamauchi wrote the initial software for Rocky III, most of which is still in use and some of which was used on Rocky IV. The rocker bogie mobility system was designed and built by Don Bickler. Jim Tran, Eddie Tunstel and Alberto Behar designed and fabricated portions of the hardware on Rocky III and IV. The authors wish to especially thank David Atkinson and Moustafa Chahine for their unfailing support.

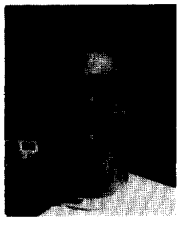
References

[1] Ronald C. Arkin, "Integrating behavioral, perceptual and world knowledge in reactive navigation," *Robotics and Autonomous Systems*, vol. 6, pp. 105–122, 1990.

[2] Bruce Bannerdt, "A broadband microseismometer for planetary applications," *Proceedings of the JPL Conference on Microtechnologies and Applications to Space Systems*, May 1992.

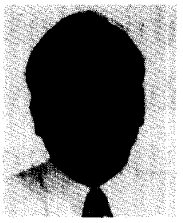
[3] Peter Bonasso, "Underwater experiments using a reactive system for autonomous vehicles," in *Proceedings of the 1991 National Conference on Artificial Intelligence*, AAAI, pp. 794–800, July 1991.

[4] Rodney A. Brooks, "A robust layered control system for a mobile robot," *IEEE Trans. Robotics Automat.*, vol. 2, no. 1, 1986.
[5] Rodney Brooks and Anita Flynn, "Fast, cheap, and out of control: A robot invasion of the solar system," *Journal of the British Interplanetary Society*, vol. 42, no. 10, pp. 478–485, 1989.
[6] Rodney Brooks, "The Behavior Language User's Guide," MIT AI Lab memo 1127, 1990.
[7] David Chapman, "Vision, instruction and action," Technical Report no. 1204, MIT AI Lab, 1990.
[8] Jonathan Connell, "A colony architecture for an artificial creature," MIT AI Tech Report 1151, 1989.
[9] Erann Gat, "ALFA: A language for programming reactive robotic control systems," in *Proceedings of the IEEE Conference on Robotics and Automation*, 1991.
[10] Erann Gat, "Robust, low-computation, sensor-driven control for task-directed navigation," in *Proceedings of the IEEE Conference on Robotics and Automation*, 1991.
[11] Erann Gat, "Integrating planning and reacting in a heterogeneous, asynchronous architecture for controlling real-world autonomous mobile robots," *Proceedings of the National Conference on Artificial Intelligence*, AAAI, 1992.
[12] Leslie Kaelbling and Stanley Rosenschein, "Action and planning in embedded agents," *Robotics and Autonomous Systems*, vol. 6, pp. 35–48, 1990.
[13] William J. Kaiser, "Microsensors and microinstruments: New measurement principles and new applications," in *Proceedings of the JPL Conference on Microtechnologies and Applications to Space Systems*, May 1992.
[14] Drew McDermott, "Transformational planning of reactive behavior," Technical Report YALEU/CSD/RR no. 941, Yale University Department of Computer Science, 1992.
[15] David P. Miller, *et al.*, "Reactive navigation through rough terrain: Experimental results," in *Proceedings of the National Conference on Artificial Intelligence*, AAAI, 1992.
[16] David Payton, "An architecture for reflexive autonomous vehicle control," in *Proceedings of the 1986 IEEE Conference on Robotics and Automation*, May 1986.
[17] David Payton, J. Kenneth Rosenblatt, and David Keirsey, "Plan guided reaction," *IEEE Trans. Syst., Man Cyber.*, vol. 20, no. 6, pp. 1370–1382, 1990.
[18] J. Kenneth Rosenblatt and David Payton, "A fine-grained alternative to the subsumption architecture for mobile robot control," in *Proceedings of International Joint Conference on Neural Networks*, Washington, D.C., June, 1989.
[19] Reid Simmons, Erik Krotkov, and John Bares, "A six-legged rover for planetary exploration," in *Proceedings of Computing in Aerospace 8*, AIAA, pp. 739–747, October, 1991.
[20] Marc G. Slack, "Situationally driven local navigation for mobile robots," JPL Publication 90-17, California Institute of Technology Jet Propulsion Laboratory, April 1990.
[21] Anthony Stentz, "The Navlab system for mobile robot navigation," Ph.D. Thesis, Carnegie Mellon University, School of Computer Science, 1990.



Erann Gat received the Ph.D. degree from the Virginia Polytechnic Institute and State University in 1991. He is currently a member of the technical staff at the California Institute of Technology Jet Propulsion Laboratory, where he has been working on autonomous mobile robots since 1988. He is also a visiting associate in the Caltech Department of Mechanical Engineering.

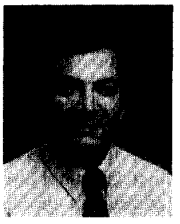
His research interests include combining strategic planning with reactive control, robot control languages, and the study of systems embedded in unmodeled environments.



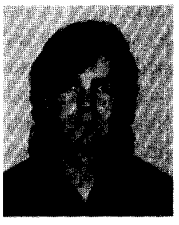
Rajiv Desai received the Bachelors degree in mechanical engineering from the Indian Institute of Technology, Masters degrees in computer engineering and mechanical engineering, and the Ph.D. degree in mechanical engineering, all from the University of Michigan, and is a candidate for the S.M. degree in management at the Sloan School of Management, Massachusetts Institute of Technology. He is currently on a sabbatical from the Jet Propulsion Laboratory, which he joined in 1988, and where he is the supervisor of the Intelligent Embedded Systems Group.

He has been involved in the development of the Rocky I-IV and Bullwinkle rover prototypes at JPL. His research interests include robot design, motion planning, behavior control, and geometric reasoning. He has over thirty publications in these areas.

In 1991, Dr. Desai was invited as a visiting scientist to the Electrotechnical Laboratory in Japan.



Robert Ivlev is a Research and Development Electronics Engineer with 20 years of experience specializing in video, RF, analog, and digital circuit design. He has spent the last eleven years working for NASA's Jet Propulsion Laboratory as a Senior Engineer Associate. He has designed hardware for numerous space shuttle missions, including the first successful flight of the Drop Physics Module on the first United States Microgravity Laboratory. He has received three NASA awards for his work and one Outstanding Performance Award from JPL. He has spent the last five years involved in robotics research, during which time he has developed electronics hardware for Rocky III and IV and for technology demonstrations of Remote Surface Inspection for the space station "Freedom."



John Loch received the Bacheors degree in aerospace engineering and mechanics from the University of Minnesota and the Master's degree in aeronautics and astronautics from the Massachusetts Institute of Technology in 1989. He is currently a member of the technical staff in the Intelligent Embedded Systems group at the Jet Propulsion Laboratory.

His major research interests are in embedded control and intelligent robotics.

David P. Miller received the Bachelor's degree in astronomy from Wesleyan University and the Ph.D. degree in computer science from Yale University in 1985. He has been an Assistant Professor at Virginia Tech and was technical group supervisor of the Robotic Intelligence Group at NASA's Jet Propulsion Laboratory, where he was the principal investigator on the Tooth and Rocky III microrovers. He is currently a Principal Scientist at the MITRE Corporation.

His major research interests are in robotic planning and system integration for a wide variety of applications.