Artificial Life and Real Robots

Rodney A. Brooks
MIT Artificial Intelligence Laboratory
545 Technology Square
Cambridge, MA 02139, USA
brooks@ai.mit.edu

"Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life," Francisco J. Varela and Paul Bourgine, eds., MIT Press, Cambridge, MA, 1992, pp. 3—10.

Abstract

The first part of this paper explores the general issues in using Artificial Life techniques to program actual mobile robots. In particular it explores the difficulties inherent in transferring programs evolved in a simulated environment to run on an actual robot. It examines the dual evolution of organism morphology and nervous systems in biology. It proposes techniques to capture some of the search space pruning that dual evolution offers in the domain of robot programming. It explores the relationship between robot morphology and program structure, and techniques for capturing regularities across this mapping.

The second part of the paper is much more specific. It proposes techniques which could allow realistic explorations concerning the evolution of programs to control physically embodied mobile robots. In particular we introduce a new abstraction for behavior-based robot programming which is specially tailored to be used with genetic programming techniques. To compete with hand coding techniques it will be necessary to automatically evolve programs that are one to two orders of magnitude more complex than those previously reported in any domain. Considerable extensions to previously reported approaches to genetic programming are necessary in order to achieve this goal.

1 Introduction

In recent years a new approach to Artificial Intelligence has developed which is based on building behavior-based programs to control situated and embodied robots in unstructured dynamically changing environments [Brooks 91c]. Rather than modularize perception, world modeling, planning, and execution, the new approach builds intelligent control systems where many individual modules directly generate some part of the behavior of the robot. In the purest form of this model each module incorporates its own perceptual, modeling, and planning requirements. An arbitration or mediation scheme, built within the framework of the modules, controls which behavior-producing module has control of which part or the robot at any given time. The programs are layered in their construction, but are non-hierarchical in their control flow, with lower levels taking care of more primitive activities and higher levels taking care of more sophisticated ones (see [Brooks 86] and [Brooks 90b] for an introduction and review, and [Brooks 91c] for a survey of the field).

This work draws its inspirations from neurobiology ethology, psychophysics, and sociology. The approach grew out of dissatisfactions with traditional robotics and Artificial Intelligence, which seemed unable to deliver real-time performance in a dynamic world. The key idea of the new approach is to advance both robotics and AI by considering the problems of building an autonomous agent that physically is an autonomous mobile robot, and that carries out some useful tasks in an environment which has not been specially structured or engineered for it.

While robots built on these principles have been demonstrated learning calibration information [Viola 90], behavior coordination [Maes and Brooks 90], and representations of the world [Mataric 90], progress in learning new behaviors has proven more difficult. Today, we are constrained to programming each new behavior by hand.

Work in Artificial Life has developed techniques for evolving programs for controlling situated, but unembodied (i.e., simulated), robots (e.g., [Langton 87], [Langton et al 90]). At some level one of the goals of Artificial Life is to move out of the digital medium into that of embodied systems. Is there a match between AL and AI? This paper explores the prospects for using Artificial Life techniques to evolve programs to control physically embodied mobile robots, so that we no longer have to do it all by hand.

There have been no reports to date of programs evolved for embodied robots. There has been work on learning new behaviors using reinforcement learning, e.g., [Kaelbling 90] and [Mahadevan and Connell 90] used Q-1earning ([Watkins 89]). The major drawback is the large number of runtime trials, many more than needed by real animals, and the need for carefully "shaping" the learning by splitting up the tasks into little pieces that the robot learns sequentially. It seems that real animals have innate built-in structures that facilitate learning particular constrained classes of behaviors. The vast numbers of trials necessary are spread over the generations, and runtime learning has a more constrained space in which it must search.

Recently [Langton 91] suggested using genetic programming for behavior-based embodied robots to overcome these limitations.

2 Genetic Programming

One way to solve the programming problem might be to use Artificial Life techniques to evolve behavior-based programs.

Previously many workers have used genetic algorithms to program software agents, typically running in cellular worlds. [Collins and Jefferson 90] is a good example. It demonstrates the evolution of both neural networks and finite state machines through a genetic algorithm running on a bit string representation.

More conventional computer programs have also been processed with genetic algorithms, such as the pioneering work of [Friedberg 58], [Friedberg et al 59], and more recently that of [Ray 90]. Robot programs, and in particular behavior-based robot programs, are much more complex than any programs that have been reported in the literature to have been so evolved. A reasonable comparison might be in terms of the memory taken to represent the programs. By this measure behavior-based robot programs axe three orders of magnitude larger than those mutated competitively by genetic techniques.

Recently, however, [Koza 90] has shown a number of stimulating results by applying genetic algorithms directly to lisp-like programs rather than to more traditional bit strings [Holland 75]. He has been very successful in a number of domains with this technique, rekindling earlier interest in the idea of mutating lisp program structures directly [Lenat 77]. [Koza 91] shows an example of synthesizing the base behaviors of [Mataric 90]'s behavior-based robot programs. He makes a number of simplifying assumptions, and reduces the search space significantly by carefully selecting the primitives by hand after examining Mataric's source code. His programs only run in simulation rather than on a physically embodied robot, but the results are nevertheless significant enough to warrant further exploration of this technique.

Before these techniques can be adopted and modified for programming physically embodied mobile robots there are a number of problems which must be addressed:

- Most likely the evolution of robot programs must be carried out on simulated robots—unfortunately there is a vast difference (which is not appreciated by people who have not used real robots) between simulated robots and physical robots and their dynamics of interaction with the environment.

- The structure of the search space of possible programs is very dependent on the representation used for programs and the primitives available to be incorporated. Careful design is necessary.

- Natural evolution co-evolved the structure of the physical entities and their neural controllers in a way which arguably cut down the size of its search space. Can some equivalent tricks be played when evolving programs for robots?

3 Simulations of Physical Robots

The number of trials needed to test individuals precludes using physical robots for testing the bulk of the control programs produced for them by genetic means. The obvious choice is to use simulated robots and then run the successful programs on the physical robots.

Previously we have been very careful to avoid using simulations ([Brooks 90b, 91b, 91c]) for two fundamental reasons.

- Without regular validation on real robots there is a great danger that much effort will go into solving problems that simply do not come up in the real world with a physical robot (or robots).

- There is a real danger (in fact, a near certainty) that programs which work well on simulated robots will completely fail on real robots because of the differences in real world sensing and actuation—it is very hard to simulate the actual dynamics of the real world.

3.1 Artifactual Problems

There has been a tendency to use cellular representations of space for simulating robots in Artificial Life. (e.g., [Langton et al 90]) and

Artificial Intelligence (e.g., [Pollack and Ringuette 90] and [Wang and Beni 90]).

These representations are good for conducting computational experiments, and help uncover many fundamental issues. Unfortunately they do not shed light on all the problems which will be encountered when using physically embodied robots. For the physical robot perspective, cellular worlds have three problems. First, there is no notion of the uncertainty that the real world presents-see the subsection below for more discussion of this point. Second, there is a tendency to not only postulate sensors which return perfect information (e.g., the cell ahead contains food—no real perception system can do such a thing), but there is a real danger of confusing the global world view and the robot's view of the world. Third, the dynamics actually tend to be more brittle than in the real world where noise and stochastic processes smooth things out quite a bit.

The dynamics mentioned in the last point often leads to problems which do not occur in the real world. A good example is being concerned with how two robots resolve a conflict when their paths must cross a single cell simultaneously. This and other equally artifactual problems are the main concern of a number of papers in this area.

But the same intellectual problem of worrying about simulated problems which do not actually appear in the real world is more general than just for cellular simulations. A number of papers have appeared that are concerned with path planning for mobile robots in nongrid worlds, i.e., in two dimensional Euclidean space. Some of these papers expend much effort on solving the problem of the paths of two robots crossing each other and introduce elaborate protocols to avoid deadlock. But real robots would never reach the state of perfect deadlock which are postulated in these papers. They would never run down their respective corridors perfectly and arrive at identical times. Simple reactive strategies would suffice to break any possible deadlock, just as random variations in ethernet controllers break deadlocks on rebroadcast.

Thus, while simulated worlds are in many ways simpler than the real world, they are paradoxically sometimes harder to operate within.

3.2 Real Worlds

For a physically embodied robot in the real world there are a number of key points to understand.

- Sensors deliver very uncertain values even in a stable world.
- The data delivered by sensors are not direct descriptions of the world as objects and their relationships.
- Commands to actuators have very uncertain effects.

A particular sensor, under ideal experimental conditions, may have a particular resolution. Suppose the sensor is a sonar. Then to measure its resolution an experiment will be set up where a return signal from the test article is sensed, and the resolution will be compared against measurements of actual distance. The experiment might be done for a number of different surface types. But when that sensor is installed on a mobile robot, situated in a cluttered, dynamically changing world, the return signals that reach the sensor may come from many possible sources. The object nearest the sensor may not be made of one of the tested materials. It may be at such an angle that the sonar pulse acts as though it were a mirror, and so the sonar sees a secondary reflection. The secondary lobes of the sonar might detect something in a cluttered situation where there was no such interference in the clean experimental situation. All sensors have comparable sets of problems associated with them. They simply do not return clean accurate readings. At best they deliver a fuzzy approximation to what they are apparently measuring, and often they return something completely different.

Even given these difficulties, sensor readings are not the same as a description of the world. Sensors measure certain quantities or indirect aspects of the world. They do separate objects from the background. They do not identify objects. They do not give pose information about objects. They do not separate out static objects, moving objects, and effects due to self motion. They do not operate in a stable coordinate system independent of the uncertain motion of the robot. They do not integrate other sensory modalities into a single consistent picture of the world. A robot operating in the real world needs a complex perceptual system. As much as 50% of the human brain seems to be devoted to perception. Off the shelf perceptual systems are not available, however. All of the problems listed above are active areas of research by perception researchers. And, as argued elsewhere [Brooks 91a, 91b], it may be impossible to treat perception as a black box with a clean interface to the rest of intelligence.

Just as sensors do not deliver simple descriptions of the world, high level action commands need many layers of refinement before they become appropriately orchestrated motor currents. In any case, the desired action and the achieved action may differ widely depending on the intricate details of the situation at hand. On flat smooth floors, odometry errors soon accumulate to the point that a a robot needs to recalibrate its position to some external reference. Besides a large unsystematic component, odometry may also have systematic aspects, e.g., depending on the relative nap of the carpet, on which the robot is operating—this has lead some researchers to try to sense the nap! The situation is much more difficult to model, of course, when the robot is in contact with an obstacle.

Sensing, and action are intimately tied together in a physical robot. They both rely on, and at the same time generate, the dynamics of the interaction of the robot with the world. Simple state space models of the world do not suffice in the internal control program of a physically embodied mobile robot.

4 Morphological Development

We now turn to the nature of the search space in which the genetic programming techniques must work.

In nature, evolution experiments with both the morphology of the individual and its neural circuitry cotemporaneously and through the same genetic mechanism. There are two important consequences of this, both of which reduce the space which evolution must search.

- The control program is evolved incrementally. Evolution is restricted initially to a small search space. The size of the space grows over many generations, but by then there is a good partial solution already found which forms the basis for searching the newer parts of the space.

- Symmetric or repeated structures naturally have symmetric or repeated neural circuits installedthey do not need to be individually evolved (e.g., a single encoding specifies the wiring of both a right and a left leg, and likewise each added segment containing a leg pair in a mutant *Drosophila* comes with neural circuitry).

Our approach to behavior-based programming of robots has always been to build layers incrementally [Brooks 86]. For genetic programming we suggest that the robot (both simulated and physical) should initially be operated with only some of its sensors, and perhaps only some of its actuators. Programs can be evolved for this simplified robot, much as our hand written programs ([Brooks 90b]) start out with layers which only use some of the capabilities of the robot. Once fundamental behaviors are present, additional sensors and actuators can be made available so that higher level behaviors can be evolved. The particular fitness function used to control the evolutionary search can be varied over time to emphasize the use of new capabilities and the need to develop higher level behaviors. There might also be some advantage to biasing the genetic operators towards operating more on the newer behaviors than on the early. (Although it certainly makes sense for the crossover operator to take pieces of old behavior eg., an orienting behavior, based on a sensor activated early in the evolutionary search would be a good prototype for an orienting behavior using a new sensor.)

In our hand written programs we capture symmetry and repeated structure by the use of macros. For example, on the six-legged robot Genghis ([Brooks 89]) there is a macro version of the leg control behaviors which gets instantiated six times, once for each leg. This suggests two ideas to again reduce the genetic search space.

- The language that is subject to genetic programming should include a macro capability. As the search learns how to use these effectively it will greatly accelerate the production of good programs.

- There must be some way to reference the symmetries and regularities of the morphological structure of the robot, in order to invoke the macros correctly. This morphological descriptor will be a constant for each particular class of robot. The constant could be evolved, but it would be much quicker to have that as an available constant somewhere in the original pool of ancestral programs.

5 Evolving Behavior-Based Programs

[Koza 91] reports on a genetic programming implementation in simulation of the navigation behaviors of [Mataric 90]'s robot Toto. The programming language he uses is a carefully chosen subset of Lisp. While adequate for the task Koza reports, based on our experience with physical robots, it is perhaps not sufficient for the more general case.

[Langton 91] has suggested the idea of genetically programming a physical robot using the Behavior Language (BL) defined in [Brooks 90a]. Unfortunately, there are many drawbacks to using BL directly.

[Lenat and Brown 84] analyze the the apparent success of the discovery system AM [Lenat 77]. They point out the crucial way in which the syntax of the Lisp programs being mutated mirrored the semantics of the world of simple mathematics concepts being explored. We can likewise expect performance of a crossover based genetic programming system to critically depend on the syntax of the language used and the way in which crossover mutates a program's semantics.

In this section we try to identify the choices made by Koza which led to his success, and compare them to what is available in BL. We propose a higher level language, called GEN, which can be compiled into BL as the target language for genetically programming physical robots.

5.1 Primitives for Genetic Programming of Robots

[Koza 90] uses a representation for Lisp programs which is crucial to the easy application of crossover. He treats an S-expression as a tree rooted with the first elements, and with one branch for each of the subsequent elements. E.g., the expression (* 1 2 3) is thought of as a tree with four nodes, a depth of two and a branching factor of three at the root node. Koza calls each thing which can occur at a non-terminal node a function, but that terminology differs from modern Lisp terminology. In fact his operators are often special forms or macros.

For instance, he might use IF-SENSOR as an operator which accesses some hidden state (or perceptual information) and depending on its value either evaluate the first argument or the second argument-thus it is not a function in the usual sense of being applied to all its evaluated arguments. Such embedded conditionals seem to be the only conditional forms used by Koza. This is for a very good reason. Pure Lisp can be thought of as a type-free language. But, in fact some values, such as the test argument to a conditional, are treated as booleans. Although everything is trivially coercible to a boolean, including a conditional requiring a boolean test would greatly increase the search space and drastically lower the density of semantically useful program trees.

In [Koza 91] careful analysis of [Mataric 90]'s original code was done in order to pick just the right set of conditionals with access to hidden state which is to be tested.

Without the hindsight from a successful implementation, GEN must allow more general testing than Koza allows. But the heuristic of embedding predicates in conditional special forms (implemented as macros) is a good one.

Koza hides critical constants in these special forms also. [Ray 90] points out the difficulty in having too many constants around in the genetic pool, and instead evolves simple constructive program segments to build them. Primitives to make this easy would be a good idea in GEN also.

5.2 Variables and Lexicality

Real BL programs use many named state variables. Sometimes these are simply used as semaphores, sometimes they are used as counters, sometimes they are used for calibration offsets, and sometimes they are used to store sensor readings, or processed sensor readings, for later comparison.

As with Lisp and BL, lexical contours should be able to be introduced at any point in a program (both languages use LET to do this). Variable names as such can not be used conveniently if crossover is to occur, as the produced programs will not be lexically meaningful very often. Instead some sort of indexing scheme is needed. A global index for a behavior is not a good idea, because then it will be hard to use crossover to duplicate little pieces of code with local variables. Instead we propose using a single form (variable-ref), and specific operators to move up and down the lexical contour tree, and to rotate around the set of variables introduced at a particular contour (often there will be just a single variable).

5.3 Depth and Breadth

In Koza's re-implementation of Toto's navigation behaviors the best solution found has 157 nodes of which 65 are terminals, and the tree has depth 12. Figure 1 shows the statistics for Mataric's original BL program for the same task. Overall it has 942 nodes, and 500 terminals, and coincidentally a maximum depth of 12.

The difference can be accounted for by three components:

1. There are a number of housekeeping operations, reporting, and debugging features included in the original BL program. These can be left out of GEN programs also.

2. The techniques used by Koza to compress the trees are not present in the BL programs. GEN will be able to get some of this benefit, but not all, as it needs to be more general than the minimal set of primitives used by Koza.

3. The code produced by Koza has only been tested in simulation, and relies a little on the simplicity of the the simulated emvironment. GEN programs for embodied robots may need to be slightly more complex than this,

It seems reasonable to assume that a general purpose GEN language program for this task might come out a factor of two to three bigger than Koza's program. Evolving a program of that size certainly seems within reach-it is not a drastic step up from Koza's results.

P	N	T	D	B	AD	AB	behavior name
3	61	25	4	4	4.8	1.6	CORRECT
3	79	35	4	6	4.5	1.7	ALIGN
2	97	41	3	5	4.9	1.7	STROLL
3	64	29	4	6	4.0	1.7	SIDES
3	96	41	4	6	5.0	1.7	VALLUMFILIA
3	24	21	2	7	1.9	7.0	PRINT-PPORT
1	10	3	4	3	2.5	1.3	GET-PPORT-DATA
1	3	1	3	2	1.7	1.0	GET-SONAR-DATA
8	230	147	5	7	3.1	2.7	BASEMON
12	229	129	6	12	3.0	2.2	BASE
1	49	28	5	9	4.2	2.3	DEBUG
40	942	500	6	12	3.8	2.0	total

Figure 1: A statistical summary of a BL program to drive the robot TOTO in its wall following. It includes various debugging behaviors and display code. The columns are P, number of processes, N, total number of nodes, T, total number of terminals, D, maximum depth, B, maximum branching factor, AD, average depth of all nodes, AB, average breadth at non-terminal nodes, and the name chosen by the programmer as the behavior name.

For comparison, figure 2 shows the statistics for a walking program for the six-legged robot Attila. Again there are a number of debugging aids and tools included in this program, along with some low level substrate processes (which need not be evolved).

5.4 Mappings

Earlier it was pointed out that there needs to be some way to relate the evolved program to the morphological regularities in the structure of the robot. There is an additional related problem which if not handled well will generate a search space with only a low density of useful program fragments.

Typically on real robots a ring of 8, say, bump sensors, or infrared proximity sensors, will be accessible as a single byte of 8 one bit values. The mapping between arrangement of those bits and the morphology of the sensors is usually quite arbitrary, and indeed there may be no natural semantically valid arrangement of the bits.

Careful attention must be paid to this issue to provide appropriate mapping primitives so that the evolving programs can pull out the right boolean bits. One simple heuristic for the robot designer is to make all such mappings consistent where possible on a particular robot design, thus maximizing the benefits for crossover of program fragments in adopting new sensors.

6 Reconnecting to Reality

Eventually the programs developed by genetic programming are to be run on physically embodied robots. There are a number of concerns in connecting the programs back to this reality.

6.1 Calibration

It is our experience that supposedly identical physical robot components are not identical. We have experienced this with very different sensing and actuator responses from supposedly identical legs on a six legged robot. We have also experienced this with 20 small wheeled robots (R-1's).

On reflecting on our past practices, we realized that when programming just a single robot system we had experimented by hand and built appropriate constants into our programs to handle the responses of our sensors and actuators. When we had the same code running on multiple copies of a robot, or robot component, we found this approach inadequate. We have found it necessary therefore to build in adaptive elements into the run-time structures of our programs which change thresholds and timeout intervals. This is not yet a formalized process, but we need to find a set of primitives that allow such adaptive elements to be constructed easily.

The implication of this for genetic programming of physical robots is clear. There must be a set of primitives available to the system so that adaptive elements can be constructed. Further, the simulation must have enough variance in it that these adaptive elements are essential to successfully run the programs on the simulated robots, so that the adaptive elements will be there when it comes time to validate on physical robots.

6.2 Adapting the Simulation

When the programs which have been evolved on the simulated robots are tried on embodied robots there will be two types of information available.

1. How well the programs work on embodied robots.

2. How different the performance of the programs is on the simulated robots and the embodied robots.

The first type of information can be used to inform the genetic programming system, as would any simulated test of the generated programs.

The second type of information should be used to tune up the simulation to better match reality. One would expect that this would be done by hand. However there is a more tantalizing possibility. Perhaps this information could be used to co-evolve the simulater using genetic programming techniques-especially if the simulator were written in the same sort of BL as the robots were programmed (our current simulator is partially written in BL). The idea would be to run the evolved robot programs on the embodied robots, then with those programs fixed, evolve the simulator until it better matched what happened in reality. There are deep issues in finding measures to compare the performance of the simulator to reality, since the robots won't be going through exactly the same sequence of operations Eventually, perhaps, this may be a viable approach.

7 Progress

At the time of writing no complete experiments have been carried out using the ideas in this paper.

We have built a simulator for multiple R-2 robots[1] It is not grid-based, but instead the coordinates of a robot can be arbitrary floating point numbers within the workspace. R-2 robots have a two wheeled differential drive with passive castors for stability. The simulator handles arbitrary independent velocities on the two wheels. There is a simple physics associated with motion of a robot when it has collided with an obstacle (which are all modeled as immovable cylinders). The sensors currently modeled are a ring of eight bump sensors, a ring of eight infrared proximity sensors, and three forward looking beacon sensors. We expect to add more sensor models. No explicit uncertainty is built into the sensor or actuator models. However, the BL program which runs the robots refers to the real time clock of the computer on which the simulation is run. Noise is therefore introduced into the system by the load on the computer, and by the interjection of the Lisp garbage collector.

The complete simulation is about 500 lines of combined Common Lisp and BL code. We have no hypothesis at this point about how well programs developed on the simulator will transfer to the real robot. This will be a critical data point in evaluating the utility of the approach.

The Behavior Language compiler has been modified so that other programs can call it, rather than having to go through the previous interface optimized for people using it to program physical robots. It already had a backend which produced a byte-code program for which a Common Lisp program provides an interpreter. Thus, BL programs can be automatically -compiled and run on a simulation machine. In addition, a higher language known as GEN has been partially built which compiles into BL programs. GEN is based on the ideas explored in the previous sections.

8 Conclusion

The key ideas presented in this paper are:

- Use genetic programming techniques to build behavior-based programs which can run real robots (this idea is due to [Langton 91]).

- Evolution and runtime adaptation are two separate issues.

- All robots will need runtime adaptation elements.

- Evolution of control structure needs to run in parallel with evolution of morphology—in robots this can be simulated by progressively enabling more sensors and actuators as layers of behaviors evolve for those already operational.

- Regularity (e.g., symmetry or repeated structures) in morphological structure should be mirrored in regularity in the control structure, and thus needs to be representable in the control language.

- Special care must be taken in design of the control language to minimize the depth and breadth of useful program trees.

- To make crossover useful in a control language with variables, a new method for representing variable references was introduced.

- There are real methodological dangers in using simulations as a testing medium in which to evolve programs which are intended eventually to run on physical robots-great care must be taken to develop a sufficient validation regime.

There are many possible approaches to using Artificial Life techniques for programming physical robots We have chosen one particular approach here,

[1] Manufactured by IS Robotics.

but many of the points of concern will be common with other approaches.

Acknowledgements

Maja Mataric, Ian Horswill and Anita Flynn provided helpful comments on drafts of this paper.

Support for this research was provided in part by the University Research Initiative under Office of Naval Research contract N00014-86-K-0685, in part by the Advanced Research Projects Agency under Office of Naval Research contract N00014-85-K-0124, in part by the Hughes Artificial Intelligence Center, and in part by Mazda Corporation.

References

[Brooks 86] "A Robust Layered Control System for a Mobile Robot", Rodney A. Brooks, *IEEE Journal of Robotics and Automation* RA-2, April, 14-23.

[Brooks 89] "Robot that Walks: Emergent Behavior from a Carefully Evolved Network", Rodney A. Brooks, *Neural Computation*, 1:2, Summer, 253-262.

[Brooks 90a] "The Behavior Language; User's Guide", Rodney A. Brooks, *MIT A.I. Lab Memo 1227*, April.

[Brooks 90b] "Elephants Don't Play Chess", Rodney A. Brooks, *in* [Maes 90], 1990, 3-15.

[Brooks 91a] "Intelligence without Representation", Rodney A. Brooks, *Artificial Intelligence* 47, Jan., 139-159.

[Brooks 91b] "Intelligence without Reason", Rodney A. Brooks, *IJCAI-91*, Sydney, Australia, Aug., 569-595,

[Brooks 91c] "New Approaches to Robotics", Rodney A. Brooks, *Science* 253, Sep., 1227-1232.

[Collins and Jefferson 90] "AntFarm: Towards Simulated Evolution", Robert J. Collins and David R. Jefferson, *in* [Langton et al 90], 579-601.

[Friedberg 58] "A Learning Machine, Part I", R. M. Friedberg, *IBM Journal of Research and Development 2, 2-13.*

[Friedberg et al 59] "A Learning Machine, Part II", R. M. Friedberg, B. Dunham, and J. H. North, *IBM Journal of Research and Development* 3, 282-287.

[Holland 75] "Adaptation in Natural and Artificial Systems", John H. Holland, *University of Michigan Press*, Ann Arbor, MI.

[Kaelbling 90] "Learning in Embedded Systems", Leslie Pack Kaelbling, *Ph.D. Thesis*, Stanford.

[Koza 90] "Evolution and Co-Evolution of Computer Programs to Control Independently-Acting Agents", John R. Koza, *Proc. First Int. Conf. on Simulation of Adaptive Behavior*, Paris, MIT Press, Cambridge, MA, 1990, 366-375.

[Koza 91] "Evolving Emergent Wall Following Robotic Behavior Using the Genetic Programming Paradigm", John R. Koza, *ECAL*, Paris, Dec.

[Langton 87] "Proceedings of Artificial Life", Christopher G. Langton (ed), *Addison-Wesley*, appeared 1989.

[Langton 91] *Personal communication*, Christopher G. Langton, September.

[Langton et al 90] "Proceedings of Artificial Life, II", Christopher G. Langton, Charles Taylor, J. Doyne Farmer, and Steen Rasmussen (eds), *Addison-Wesley*, appeared 1991.

[Lenat 77] "The Ubiquity of Discovery", Douglas B. Lenat, *IJCAI-77*, Cambridge, MA, Aug., 1093-1105.

[Lenat and Brown 84] "Why AM and EURISKO Appear to Work", Douglas B. Lenat and John Seely Brown, *Artificial Intelligence* 23,269-294.

[Maes 90] "Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back", Pattie Maes (ed), *MIT Press*, Cambridge, MA, 1990.

[Maes and Brooks 90] "Learning to Coordinate Behaviors", Pattie Maes and Rodney A. Brooks, *AAAI-90*, Boston, MA, 1990, 796-802.

[Mahadevan and Connell 90] "Automatic Programming of Behavior-based Robots using Reinforcement Learning", Sridhar Mahadevan and Jonathan Connell, *IBM T.J. Watson Research Report*, Dec.

[Mataric 90] "A Distributed Model for Mobile Robot Environment-Learning and Navigation", Maja J. Mataric, *MIT A.I. Lab Technical Report 1228*, May.

[Pollack and Ringuette 90] "Introducing the Tileworld: Experimentally Evaluating Agent Architectures", Martha E. Pollack and Marc Ringuette, *AAAI-90*, Boston, MA, August, 183-189.

[Ray 90] "An Approach to the Synthesis of Life", Thomas S. Ray, *in* [Langton et al 90], 371-408.

[Viola 90] "Adaptive Gaze Control", Paul A. Viola, *MIT SM Thesis*, 1990.

[Wang and Beni 90] "Distributed Computing Problems in Cellular Robotic Systems", Jing Wang and Gerardo

Beni, *IEEE/RSJ International Workshop on Intelligent Robots and Systems*, Ikabara, Japan, 819-826.

[Watkins 89] “Learning from Delayed Rewards”, Christopher Watkins, *Ph.D. Thesis*, King's College, Cambridge.

P	N	T	D	B	AD	AB	behavior name
6	470	258	6	21	4.1	2.2	MONITOR
2	39	25	5	4	3.2	2.6	PARAMS
2	55	28	4	5	5.5	2.0	R3-FIND
2	55	28	4	5	5.5	2.0	R2-FIND
2	55	28	4	5	5.5	2.0	R1-FIND
2	55	28	4	5	5.5	2.0	L3-FIND
2	55	28	4	5	5.5	2.0	L2-FIND
2	55	28	4	5	5.5	2.0	L1-FIND
3	61	34	5	9	3.8	2.1	STANDUP
1	3	2	2	2	1.7	2.0	ROLLER
2	8	4	3	2	2.0	1.5	R3-SLEG
2	8	4	3	2	2.0	1.5	R2-SLEG
2	8	4	3	2	2.0	1.5	R1-SLEG
2	8	4	3	2	2.0	1.5	L3-SLEG
2	8	4	3	2	2.0	1.5	L2-SLEG
2	8	4	3	2	2.0	1.5	L1-SLEG
6	101	51	9	9	5.1	1.9	FLIPPER
2	34	13	5	3	3.9	1.5	FLIPPED?
1	23	13	3	8	2.6	2.2	CALIBRATE
17	485	224	5	10	3.8	1.8	R3-ALEG
17	485	224	5	10	3.8	1.8	R2-ALEG
17	485	224	5	10	3.8	1.8	R1-ALEG
17	485	224	5	10	3.8	1.8	L3-ALEG
17	485	224	5	10	3.8	1.8	L2-ALEG
17	485	224	5	10	3.8	1.8	L1-ALEG
2	20	9	5	2	3.2	1.6	TILTER
2	11	6	3	2	2.4	1.8	TILTINTERFACE
2	123	53	3	17	6.2	1.7	STATIC-CALIBRATE
3	21	12	4	2	2.8	2.0	PANIC-MAINTAIN
9	151	78	5	17	3.0	1.9	HEADER
2	65	26	5	8	3.8	1.6	SLEEP
4	24	14	4	2	2.7	2.0	DROWSINESS-MAINTAIN
3	61	33	6	10	3.3	2.1	I2CBUS
2	31	13	8	2	5.5	1.6	LISTEN-HOST
176	4526	2176	9	21	4.0	1.9	total

Figure 2: A statistical summary of a walking program for the robot Attila. The columns have the same labels as those in figure 1. Notice that some behaviors are duplicated six times-once for each leg of the robot.