From: AAAI-90 Proceedings. Copyright ©1990, AAAI (www.aaai.org). All rights reserved.

Learning to Coordinate Behaviors

Pattie Maes & Rodney A. Brooks
AI-Laboratory
Massachusetts Institute of Technology
545 Technology Square
Cambridge, MA 02139
pattie@ai.mit.edu
brooks@ai.mit.edu

Abstract

We describe an algorithm which allows a behavior-based robot to learn on the basis of positive and negative feedback when to activate its behaviors. In accordance with the philosophy of behavior-based robots, the algorithm is completely distributed: each of the behaviors independently tries to find out (i) whether it is *relevant* (i.e. whether it is at all correlated to positive feedback) and (ii) what the conditions are under which it becomes *reliable* (i.e. the conditions under which it maximizes the probability of receiving positive feedback and minimizes the probability of receiving negative feedback). The algorithm has been tested successfully on an autonomous 6-legged robot which had to learn how to coordinate its legs so as to walk forward.

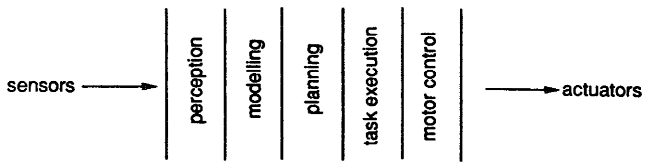

Figure 1: Classical decomposition of an autonomous robot.

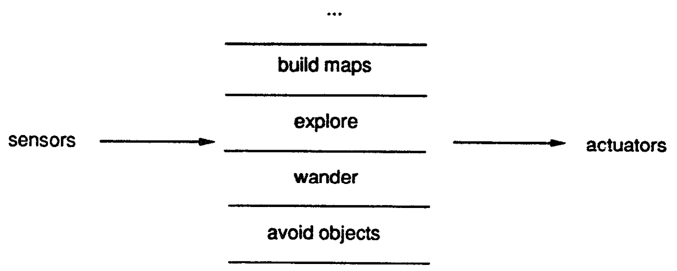

Figure 2: Behavior-based decomposition of an autonomous robot.

Situation of the Problem

Since 1985, the MIT Mobile Robot group has advocated a radically different architecture for autonomous intelligent agents (Brooks, 1986). Instead of decomposing the architecture into functional modules, such as perception, modeling, and planning (figure 1), the architecture is decomposed into task-achieving modules, also called *behaviors* (figure 2). This novel approach has already demonstrated to be very successful and similar approaches have become more widely adopted (cfr. for example (Brooks, 1990) (Rosenschein & Kaelbling, 1986) (Arkin, 1987) (Payton, 1986) (Anderson & Donath, 1988) (Yamaushi, 1990) (Zhang, 1989)).

One of the main difficulties of this new approach lies in the control of behaviors. Somehow it has to be decided which of the behaviors should be active and get control over the actuators at a particular point in time. Until now, this problem was solved by precompiling the control flow and priorities among behaviors either by hand (Brooks, 1986), or automatically, using a description of the desired behavior selection (Rosenschein & Kaelbling, 1986). In both cases the result is some "switching circuitry" among the behaviors which is completely fixed at compile time by the designer.

However, for more complicated robots prewiring such a solution becomes either too difficult or impractical. As the number of behaviors goes up, the problem of control and coordination becomes increasingly complex. Additionally, it is often too difficult for the programmer to fully grasp the peculiarities of the task and environment, so as to be able to specify what will make the robot successfully achieve the task (Maes, 1990).

We therefore started developing an algorithm for learning the control of behaviors through experience. In accordance with the philosophy of behavior-based robots, the learning algorithm is completely distributed. There is no central learning component, but instead each behavior tries to learn when it should become active. It does so by (i) trying to find out what the conditions are under which it maximizes positive feedback and minimizes negative feedback, and (ii) measuring how relevant it is to the global task (whether it is correlated to positive feedback).

We hope that ultimately, this learning algorithm will allow us to program the behavior of a robot by selecting a number of behaviors from a library of behaviors, connecting these to the actual sensors and actuators on the robot, defining positive and negative feedback functions, and making each of the behaviors learn from experience when it is appropriate for it to become active.

The Learning Task

The learning task we are concerned with is defined as follows. Given a robot which has:

- a vector of binary perceptual conditions which are either being perceived (or "on") or not being perceived (or "off") at every instant of time,
- a set of behaviors; where a behavior is a set of processes involving sensing and action; a behavior has a precondition list, which is a conjunction of predicates testing a specific value (on or off) for a certain perceptual condition; a behavior may become active when all of its preconditions are fulfilled; an active behavior executes its processes,
- a positive feedback generator, which is binary and global, i.e. at every time t, the robot (and therefore all of the behaviors) either receive positive feedback or not,
- a negative feedback generator, which is again binary and global.

The learning task is to incrementally change the precondition list of behaviors so that gradually only those behaviors become active that fulfill the following two constraints:

1. they are *relevant*,
   where a relevant behavior is a behavior that is positively correlated to positive feedback (i.e. positive feedback is more often received when the behavior is active then when it is is not active) and not positively correlated to negative feedback (i.e. either not correlated at all or inversely correlated),
2. they are *reliable*,
   where a reliable behavior is defined as a behavior that receives consistent feedback (i.e. the probability of receiving positive (respectively negative) feedback when the behavior is active is close enough to either 1 or 0).

An additional requirement, imposed by our philosophy, is that we want the algorithm to be *distributed.* It should allow individual behaviors to change their precondition list in response to certain feedback patterns so that the global behavior of the set of behaviors converges towards a situation where maximal positive feedback and minimal negative feedback is received. Finally, three additional constraints are related to the fact that this algorithm has to be useful for real robots in unconstrained environments: (i) the algorithm should be able to deal with *noise*, (ii) the algorithm should be *computationally inexpensive*, so that it can be used in real-time, and (iii) the algorithm should support *readaptation*, i.e. if the robot changes (e.g. some component breaks down), or its environment changes (e.g. the feedback generators change) the algorithm will make the robot adapt to this new situation (which possibly involves forgetting or revising learned knowledge).

We adopted some simplifying, but realistic, assumptions. One is that, for every behavior, there exists at least one conjunction of preconditions for which the probability of positive feedback as well as the probability of negative feedback are within some boundary (which is a parameter of the algorithm) from either 0 or 1. Another important assumption is that feedback is immediate (there is no delayed feedback) and does not involve action sequences. And finally only conjunctions of conditions (including negations) can be learned. The last section of the paper sketches how the algorithm could be extended to deal with the more general problem[1]. Nevertheless, even after adopting these simplifying assumptions, the learning task is still far from trivial. More specifically, the global search space for a robot with n behaviors, and m binary perceptual conditions is $n * 3^m$, since every behavior possibly has to learn an "on", "off" or "don't-care" value for every perceptual condition.

The Algorithm

The learning algorithm employed by each behavior is the following. Each behavior starts from a "minimal" precondition list. More specifically, only conditions that are necessary in order to be able to execute the processes of the behavior are present. Behaviors maintain data about their performance. A first set of data is related to whether the behavior is relevant or not. A behavior measures i, j, k and l:

	active	not active
positive feedback	j	k
no positive feedback	l	m

Where, j is the number of times positive feedback happened when the behavior was active, k is the number of times positive feedback happened when the behavior was not active, l is the number of times positive feedback did not happen when the behavior was active, and m is the number of times negative feedback did not happen when the behavior was not active. The same statistics are maintained for negative feedback. The statistics are initialized at some value N (for example $N = 10$) and "decayed" by multiplying them with $\frac{N}{N+1}$ every time they are updated. This ensures that impact of past experiences on the statistics is less than that of more recent experiences. The *correlation* (the Pearson product-moment correlation coefficient) between positive feedback and the status of the behavior is defined as

$$corr(P, A) = \frac{j * m - l * k}{\sqrt{(m + l) * (m + k) * (j + k) * (j + l)}}$$

[1] Some additional assumptions being made are that the robot is first of all, able to do experiments, and second, that these do not involve too big risks (the environment can not be too hostile nor the robot too fragile).

This gives a statistical measure of the degree to which the status of the behavior (active or not active) is correlated with positive feedback happening or not. $corr(P, A)$ ranges from -1 to 1, where a value close to -1 represents a negative correlation (feedback is less likely when the behavior is active), 0 represents no correlation and 1 represents a positive correlation (feedback is more likely when the behavior is active). In an similar way, $corr(N, A)$, i.e. the correlation between the status of the behavior and negative feedback is defined. The *relevance* of a particular behavior is defined as

$$corr(P, A) - corr(N, A)$$

It is used by the algorithm to determine the probability that the behavior will become active (which is related to the effort which will be put into doing experiments in order to improve the behavior, i.e. making it more reliable). The relevance of a behavior ranges from -2 to +2. The more relevant a behavior is, the more chance it has of becoming active. A behavior that is not very relevant has little chance of becoming active. So, these statistics makes it possible to determine which behaviors are the interesting ones (relevant ones). The relevant behaviors are not necessarily very reliable yet: they might only receive positive feedback in a minority of the times they are active. They also might still cause a lot of negative feedback to be received. All that is "known" is that positive feedback will be more likely received when these behaviors are active, than when they are not active (respectively negative feedback being less likely). The *reliability* of a behavior is defined as (where index P stands for positive feedback and index N stands for negative feedback)

$$min(max(\frac{j_P}{j_P + l_P}, \frac{l_P}{j_P + l_P}), max(\frac{j_N}{j_N + l_N}, \frac{l_N}{j_N + l_N}))$$

The reliability of a behavior ranges from 0 to 1. When the value is close to 1, the behavior is considered very reliable (i.e. the feedback is very consistent: the probability of receiving feedback is either close to 0 or to 1). The reliability of a behavior is used by the algorithm to decide whether the behavior should try to improve itself (i.e. learn more conditions or modify the existing preconditions). If the behavior is not reliable enough, i.e. if either the negative feedback is inconsistent or the positive feedback is inconsistent or both, then one or more additional preconditions are relevant. In this case, the behavior will pick a new perceptual condition to monitor in order to determine whether this condition might be related to the inconsistent feedback [2]. An additional set of statistics is related to the specific condition being monitored (if there is one):

	cond. on	cond. off
positive feedback	n	o
no positive feedback	p	q

Where n is the number of times positive feedback happened when the behavior was active and the condition was on, o is the number of times positive feedback happened when the behavior was active and the condition was off (not on), p the number of times positive feedback did not happen when the behavior was active and the condition was on, and q is the number of times negative feedback did not happen when the behavior was active and the condition was off. Again the same statistics are maintained for negative feedback. If the behavior notices a strong correlation between the condition being monitored and positive and/or negative feedback, it will adopt this condition as a new precondition. More specifically, if the correlation

$$corr(P, on) = \frac{n * q - p * o}{\sqrt{(q + p) * (q + o) * (n + o) * (n + p)}}$$

becomes close to 1 (respectively -1), then the condition will be adopted in the precondition list, with a desired value of "on" (respectively "off"). And similarly, if the correlation for negative feedback becomes close to 1 (respectively -1), then the condition will be adopted in the precondition list, with a desired value of "off" (respectively "on"). If the values for positive and negative feedback are incompatible, the one suggested by the negative feedback dominates. From the moment a new condition has been learned, a behavior only becomes active when this condition has the desired value. If after monitoring a condition for a while, the behavior doesn't notice any correlation between the value of the condition and positive or negative feedback, and the behavior is still not reliable enough, it will start monitoring another condition.

After learning a new condition, the behavior will not necessarily be completely reliable. There might still be other conditions related to the feedback. Until the behavior is reliable enough, it will try to find extra preconditions [3]. Notice that the list of conditions being monitored/evaluated is circular. When all of the conditions have been evaluated and feedback is still inconsistent, the behavior will start monitoring conditions from the start of the list again, reevaluating also those conditions which have already been taken up in the precondition list. A behavior might "forget" something it learned and reevaluate the relevance of that condition. This guarantees that if the environment (e.g. the feedback) or the robot changes, the behaviors are able to adapt to the new situation.

The control strategy of the algorithm is as follows. Behaviors are grouped into groups which control the same actuators. At every timestep the selectable behaviors in every group are determined (those behaviors which are

[2] Currently there is a complete connectivity between behaviors and perceptual conditions. The connectivity will be decreased in a subsequent implementation through the use of a switchboard. In applications involving a large vector of perceptual conditions, one could restrict the subset of conditions that a particular behavior considers for learning.

[3] We make the assumption here that for every condition that has to be learned the correlation to feedback is independently detectable. This is likely to be true, because we are not dealing with the problem of learning disjunctions.

not yet active and whose preconditions are fulfilled). For each of these groups, one or zero behaviors are selected probabilistically according to (in order of importance)
• the relative relevance of behaviors,
• the reliability of behaviors, and
• the "interestingness" of the current situation for behaviors, where a situation is more interesting for a behavior if the condition being monitored by the behavior appears in the situation with a value (on or off) that has been experienced a lesser number of times.

The selected behaviors are then activated. The probabilistic nature of the selection process ensures that there is a balance between behaviors being selected (i) because they are successful (are relevant and reliable) and (ii) because of experimentation purposes (to learn). Notice that the learning algorithm is biased: if behaviors are not very relevant (in comparison with other behaviors) they have very little chance of becoming active, which means that little effort is put into making them more reliable (learn new preconditions).

Finally, there are a number of global parameters which can be varied to change the algorithm
• how strong a condition has to be correlated to adopt it as a new precondition,
• how long a condition is monitored before it is dropped,
• how reliable a behavior should try to become,
• how adaptive the behavior is (the relative importance of new data versus data of past experiences).

These parameters have to be tuned to the particular circumstances of task and robot at hand.

A Robot that Learns to Walk

The Task

The described algorithm is being tested on a six-legged robot, called Genghis (see figures 4 and 5). The goal of the experiment is to make Genghis learn to walk forward. This task was chosen because of its complexity. The current version consists of 12 behaviors learning about 6 perceptual conditions, which corresponds to a search space of $12 * 3^6 = 8748$ nodes. Another reason for choosing this task was the availability of a 6-legged robot with a lot of degrees of freedom (12 to be precise) (Angle, 1989). The final reason is that a lot is known both about insect walking (Wilson, 1966) (Beer, 1989) and about 6-legged robot walking (Donner, 1987) (Brooks, 1989). The results reported in this literature demonstrated that the task was feasible (that the completely distributed walking which would result from our learning was indeed robust and successful). This literature also made it possible to compare and interpret our results.

The Robot

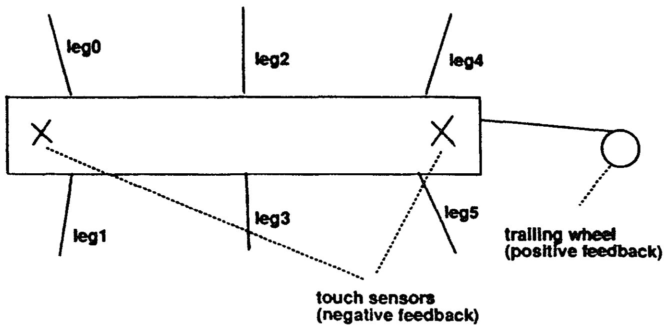

Figure 3: Schematic representation of Genghis, its positive and negative feedback sensors and its distributed collection of learning behaviors.

Genghis is an autonomous six-legged robot with twelve servo motors controlling the two degree of freedom legs (Angle, 1989). It has 4 on board 8-bit microprocessors linked by a 62.5Kbaud token ring. The total memory usage of the robot is about 32Kbytes. Genghis has been programmed before to walk over rough terrain and follow a person passively sensed in the infrared spectrum (Brooks, 1989). Our experiments were programmed using the Behavior Language and Subsumption Compiler (Brooks, 1989b). The entire learning program runs on board.

The sensors used in this experiment are two touch sensors on the bottom of the robot (one in the front and one in the back) and a trailing wheel which measures forward movement. Negative feedback is received by all of the behaviors every time at least one of the touch sensors fires. Positive feedback is received every time the wheel measures forward movement. We equipped Genghis with a dozen behaviors: 6 swing-leg-forward behaviors (which move a leg that is backward, up, forward and then down again), 6 swing-leg-backward behaviors (which move a leg that is forward, up, backward and then down again) (figure 3). Further there is one horizontal balance behavior, which sums the horizontal angles of the legs and sends a correction to all of the legs so as to reduce that sum to 0 (i.e. if one leg is moved forward, all of the legs are moved backwards a little). The 12 swing behaviors try to learn what the conditions are under which they should become active. The vector of binary perceptual conditions has 6 elements, each of which records whether a specific leg is up (not touching the ground).

Results

In a first experiment only six swing forward behaviors plus the balance behavior were involved. Genghis was able to learn to activate behaviors safely (avoiding negative feedback or "falling on its belly") and successfully (producing positive feedback or moving forward). More specifically, it learned to adopt a tripod gait, keeping three legs on the ground at any moment: the middle leg on one side and the front and back leg on the other side. Notice that this task is not trivial: negative feedback is not related to particular behaviors (none of the behaviors by itself causes negative feedback), but rather to the way they are coordinated (or uncoordinated). It is therefore a necessity in this application that actions are explored by the robot in parallel. This extra difficulty is successfully handled by the algorithm: the distributed learning behaviors are able to learn a task which requires their coordination.

This experiment has been successfully demonstrated on the robot. Using a non-intelligent search through the condition vector (i.e. every behavior monitors the conditions in the same order, starting with the status of the first leg, then the status of the second leg, etc) this takes in the order of 10 minutes. Using an intelligent search (i.e. every behavior starts by monitoring the status of legs that are nearby, then the ones that are further away, and so on), this experiment takes approximately 1 minute and 45 seconds.

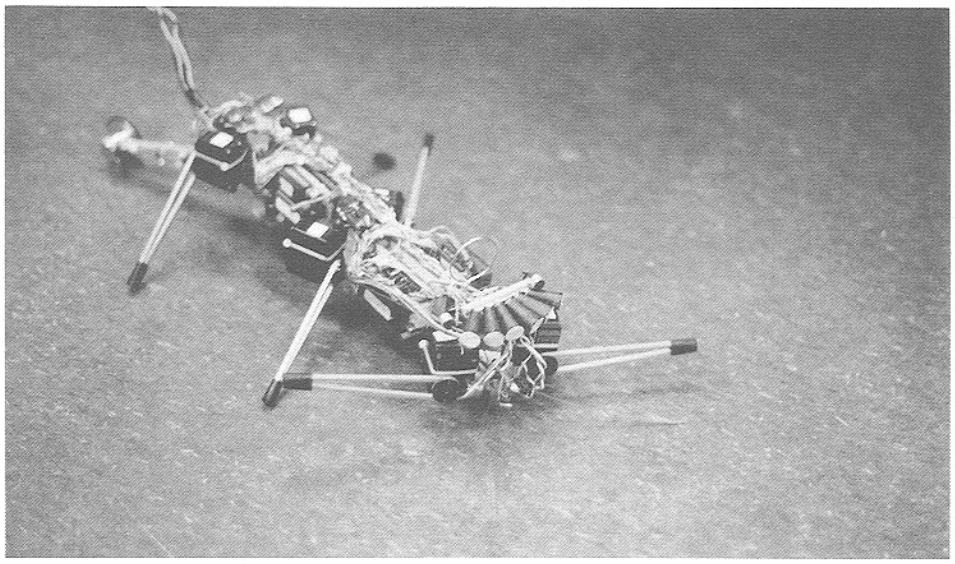
Figure 4: Genghis learning to walk. Initially, the behaviors do not know yet how they are supposed to coordinate (under what conditions they should become active). Since the two front leg "swing forward" behaviors are being activated at the same time, Genghis falls down and receives negative feedback from the touch sensor mounted on the front of its belly.

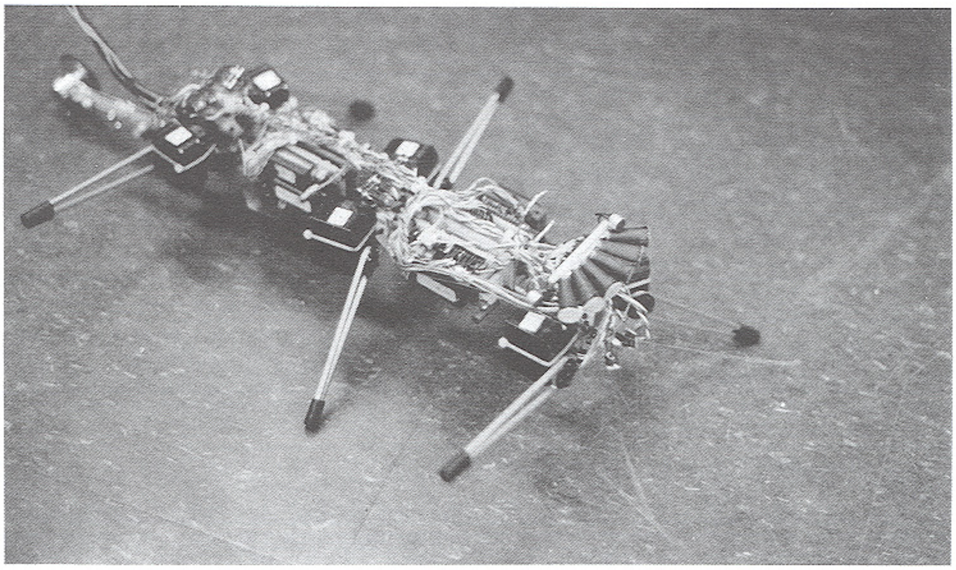
Figure 5: Gradually, a more coherent "walk" emerges. In this case the global behavior converges towards a tripod gait: two groups of three legs are being swung forward alternately.

The reason why this is so much faster is that in this case, behaviors learn and converge more or less simultaneously (in the non-intelligent search case all of the behaviors have to "wait" for leg 4 and 5 to go through the whole condition vector before finding correlated conditions). The preconditions that are learned are that a swing-forward behavior is only allowed to become active when the neighboring legs (e.g. leg3 and leg0 in the case of leg1) are down. Actually, we noticed that not all of the behaviors learn that both of the neighboring legs have to be down. If for example leg0 and leg4 learned not to be active at the same time as leg2, then leg2 doesn't have to learn how to avoid negative feedback, because its two neighbors are taking care of the coordination problem.

In a second experiment, which has been demonstrated in simulation (it is therefore difficult to compare the resulting convergence times), six swing-backward behaviors were added. The robot now also had to learn that only certain behaviors are relevant for receiving positive feedback (in the first experiment, positive feedback didn't play much of a role, because every behavior was correlated to positive feedback under all conditions). More specifically, the robot had to learn that even though the swing-leg-backward behaviors do not cause negative feedback to be received (when coordinated), they should never become active because they are not correlated to positive feedback. In our simulation, the "non-relevant" swing-backward behaviors slowly die out, because they are not correlated to positive feedback. Their probability of becoming active gradually goes down, so that they have less opportunities to find out what the conditions are under which they can minimize negative feedback. Most of them "die out" before they are able to find the optimal list of preconditions so as to avoid negative feedback.

The gait that emerges in the experiments is the *tripod gait*, in which alternatively 2 sets of 3 legs are simultaneously swung forward. As reported by Wilson (Wilson, 1966) and confirmed in simulations by Beer (Beer, 1989), the gait that emerges in a distributed 6-legged walking creature is an emergent property of the time it takes to push a leg backwards during the "stance phase". One of the experiments we are working on right now is to try to obtain different gaits as a result of varying the speed of the stance phase and by disconnecting one of the legs at run time. The circular monitoring scheme should take care that the behaviors can adapt to this new situation and modify their precondition lists accordingly. Another experiment we are currently working on is to make Genghis learn to walk backward, by adding a switch which inverts the movement feedback.

Related Learning Work

This work is related to Drescher's PhD thesis on "translating Piaget into LISP" (Drescher, 1989). The main differences are: that our behaviors (corresponding to his "schemas") do not maintain statistics for all of the perceptual conditions in the environment, but instead only for one or zero conditions at the time. As a consequence our algorithm is less computationally complex. Drescher's algorithm would not be usable in real time. A second important difference is that the algorithms are concerned with different learning tasks. In Drescher's case the task is to discover the regularities of the world when taking actions (a condition-action-effect kind of representation of the world is built up), while in the work presented here the only things learned are the conditions which optimize positive feedback and minimize negative feedback. Our system evolves towards a task-dependent (goal-oriented) solution, while in Drescher's case, generally useful knowledge is built up.

There is further also some relation to Classifier Systems and Genetic Algorithms (Holland et al., 1986) (for an application to control problems cfr. (Greffenstette, 1989)). The main difference is that our learning technique is basically constructivist, while theirs is selectionist. In our algorithm the right representations are built up (in an incremental way) instead of being selected. An advantage is that our algorithm is faster because it does not perform a "blind", unstructured search. It further also uses memory more efficiently because there is no duplication of information (all the information about one action is grouped in one behavior) and because we only monitor/explore a certain condition when there is a need for it (the behavior is not reliable yet).

Finally, the problem studied here is related to a class of algorithms called Reinforcement Learning Algorithms (Sutton, 1984)(Sutton, 1988)(Sutton, 1990)(Kaelbling, - 1990)(also related are (Narendra & Thathachar, 1989) and (Berry & Fristedt, 1985)). The main differences are that (i) the algorithm discussed here is distributed and parallel: several actions an be taken at once, and asynchronously (this is even crucial to learn the tripod gait, for example), (ii) this algorithm is action- oriented, whereas reinforcement learning algorithms are state-oriented (utility functions are associated with states, while here relevance and reliability are associated with actions) and (iii) here feedback is binary and dual (positive and negative), whereas in reinforcement learning the utility function is real valued (both have advantages and disadvantages: the former's advantage is that positive and negative goals/feedback are treated separately, while the latter's advantages is that there is a more gradual evaluation).

Conclusion and Future Work

We have developed a learning algorithm which allows a behavior-based robot to learn when its behaviors should become active using positive and negative feedback. We tested the algorithm by successfully teaching a 6-legged robot to walk forward. In future work we plan to test the generality of the algorithm by doing more experiments with the same and different robots for different tasks and by studying the properties and limitations of the algorithm with a mathematical model.

We further plan some extensions to the algorithm, the first one being the addition of a mechanism to deal with delayed feedback (or learning action sequences). Three possible solutions to this problem will be investigated: (i) the usage of some Bucket Brigade Algorithm or Temporal Difference method (Sutton, 1988) (Holland et al., 1986), (ii) the extension of the perceptual condition vector with conditions representing the past actions taken and (iii) composing actions into macro-actions (so that feedback is still immediate for such an action sequence).

Acknowledgements

Grinell Moore did most of the mechanical design and fabrication of Genghis. Colin Angle did much of the processor design and most of the electrical fabrication. Olaf Bleck built the sensors providing positive and negative feedback and replaced the broken servos every time our buggy programs "tortured" the robot. Leslie Kaelbling, Maja Mataric and Paul Viola provided useful comments on an earlier draft of this paper. Richard Lathrop and David Clemens helped with the statistics.

Supported by Siemens with additional support from the University Research Initiative under Office of Naval Research contract N00014-86-K-0685, and the Defense Advanced Research Projects Agency under Office of Naval Research contract N00014-85-K-0124. The first author is a research associate of the Belgian National Science Foundation. She currently holds a position as visiting professor at the M.I.T. Artificial Intelligence Laboratory.

Bibliography

Anderson T.L. and Donath M. (1988) A computational structure for enforcing reactive behavior in a mobile robot. In: Mobile Robots III, Proc. of the SPIE conference, Vol. 1007, Cambridge, MA.

Angle C.M. (1989) Genghis, a six-legged autonomous walking robot. Bachelors thesis, Department of EECS, MIT.

Arkin R. (1987) Motor schema based navigation for a mobile robot: An approach to programming by behavior. IEEE Conference on Robotics and Automation '87.

Beer R.D. (1989) Intelligence as Adaptive Behavior: An Experiment in Computational Neuroethology. Technical Report 89-118, Center for Automation and Intelligent Systems Research, Case Western Reserve University.

Berry D.A. and Fristedt B. (1985) Bandit problems: Sequential allocation of experiments. Chapman and Hall, London.

Brooks R.A. (1986) A robust layered control system for a mobile robot. IEEE Journal of Robotics and Automation. Volume 2, Number 1.

Brooks R.A. (1989) A robot that walks: Emergent behavior from a carefully evolved network. Neural Computation, 1(2).

Brooks R.A. (1989b) The behavior Language; User's - Guide. Implementation Note. AI-laboratory, MIT.

Brooks R.A. (1990) Elephants don't play chess. In: P. Maes (ed.) Designing Autonomous Agents, Bradford-MIT Press, in press. Also: special issue of Journal of Robotics and Autonomous Systems, Spring '90, North-Holland.

Donner M.D. (1987) Real-time control of walking. Progress in Computer Science series, Vol. 7, Birkhauser, Boston.

Drescher G.L. (1989) Made-Up Minds: A Constructivist Approach to Artificial Intelligence, PhD Thesis, Department of EECS, MIT.

Greffenstette J.J. (1989) Incremental learning of control strategies with genetic algorithms. Proceedings of the - Sixth International Workshop on Machine Learning, Morgan Kaufmann.

Holland J.H., Holyoak K.J. Nisbett R.E. and Thagard P.R. (1986) Induction: Processes of inference, learning and discovery. MIT-Press, Cambridge, MA.

Kaelbling L. (1990) Learning in embedded systems, PhD thesis, Stanford Computer Science Department, forthcoming.

Maes P. (1990) Situated agents can have goals. In: P. Maes (ed.) Designing Autonomous Agents, Bradford-MIT Press, in press. Also: special issue of Journal of Robotics and Autonomous Systems, Spring '90, North-Holland.

Narendra K, and Thathachar M.A.L. (1989) Learning Automata, an Introduction. Prentice Hall, New Jersey.

Payton D.W. (1986) An architecture for reflexive autonomous vehicle control. IEEE Robotics and Automation Conference '86, San Francisco.

Rosenschein S.J. and Kaelbling L. (1986) The synthesis of digital machines with provable epistemic properties, in Joseph Halpern, ed, Proceedings of the Conference on Theoretical Aspects of Reasoning About Knowledge, Monterey, CA.

Schlimmer J.C. (1986) Tracking Concept Drift, Proceedings of the Sixth National Conference on Artificial Intelligence '86.

Sutton R. (1984) Temporal credit assignment in reinforcement learning. Doctoral dissertation, Department of Computer and Information Science, University of Massachusetts, Amherst.

Sutton R. (1988) Learning to predict by the methods of Temporal Differences. Machine Learning Journal, Vol. 3, 9-44.

Sutton R. (1990) Integrated architectures for learning, planning and reacting based on approximating dynamic programming. Proceedings of the Seventh International Conference on Machine Learning.

Wilson D.M. (1966) Insect walking. Annual Review of Entomology, 11: 103-121.

Yamaushi B. (1990) Independent Agents: A Behavior-Based Architecture for Autonomous Robots. Proceedings of the Fifth Annual SUNY Buffalo Graduate Conference on Computer Science '90, Buffalo, NY.

Zhang, Y. (1989) Transputer-based Behavioral Module for Multi-Sensory Robot Control. 1st International Conference in Artificial Intelligence and Communication Process Architecture '89, London, UK.