**School of Informatics, University of Edinburgh**

**Institute of Perception, Action and Behaviour**

# Mobile Robot Control based on Boolean Logic with Internal Memory

by

DaeEun Kim, John Hallam

**Informatics Research Report EDI-INF-RR-0214**

**School of Informatics** **June 2004**
http://www.informatics.ed.ac.uk/

# Mobile Robot Control based on Boolean Logic with Internal Memory

DaeEun Kim, John Hallam

Informatics Research Report EDI-INF-RR-0214

SCHOOL *of* INFORMATICS
Institute of Perception, Action and Behaviour

June 2004

Final Version appears at Advances in Artificial Life, Lecture Notes in Computer Science vol. 2159 Springer-Verlag, 2001

**Abstract :**

The purpose of this paper is to explore the effect of adding known amounts of memory to pure reactive systems in a variety of tasks. Using a finite state machine approach, we construct controllers for a simulated robot for five tasks — obstable avoidance, wall following, exploration, and box pushing—with two sensor configurations using evolutionary computation techniques, and compare the performance of stateless and memory-based controllers. For obstacle avoidance and exploration no significant difference is observed; for wall-following and box pushing, stateless controllers are significantly worse than memory-based but increasing amounts of memory give no significant increase in performance. The need for memory in these cases reflects a need to discriminate sensorimotor contexts to effectively perform the task.

**Keywords** :

Copyright © 2004 by The University of Edinburgh. All Rights Reserved

The authors and the University of Edinburgh retain the right to reproduce and publish this paper for non-commercial purposes.

Permission is granted for this report to be reproduced by others for non-commercial purposes as long as this copyright notice is reprinted in full in any reproduction. Applications to make other use of the material should be addressed in the first instance to Copyright Permissions, School of Informatics, The University of Edinburgh, 2 Buccleuch Place, Edinburgh EH8 9LW, Scotland.

# Mobile Robot Control based on Boolean Logic with Internal Memory

DaeEun Kim and John C.T. Hallam

Institute of Perception, Action, and Behavior
Division of Informatics, 5 Forrest Hill
The University of Edinburgh
United Kingdom
{daeeun,john}@dai.ed.ac.uk

**Abstract**

The purpose of this paper is to explore the effect of adding known amounts of memory to pure reactive systems in a variety of tasks. Using a finite state machine approach, we construct controllers for a simulated robot for five tasks—obstacle avoidance, wall following, exploration, and box pushing—with two sensor configurations using evolutionary computation techniques, and compare the performance of stateless and memory-based controllers. For obstacle avoidance and exploration no significant difference is observed; for wall-following and box pushing, stateless controllers are significantly worse than memory-based but increasing amounts of memory give no significant increase in performance. The need for memory in these cases reflects a need to discriminate sensorimotor contexts to effectively perform the task.

## 1 Introduction

The purpose of this paper is to explore the effect of adding known amounts of memory to pure reactive systems in a variety of tasks. The memory elements will be represented in the form of a set of rules with states in our approach, equivalent to a Boolean Logic Network with a fixed number of 1-bit registers. First, we discuss the way memory is generally implemented in behavior-based robotics systems. Then we describe a set of experiments that reveal behavioral and performance differences between a pure reactive controller and a memory-based reactive controller evolved for various particular tasks, and where appropriate we explain the effect of memory on the resulting robot behaviors.

Reactive systems in mobile robots are often seen as a direct coupling between perception and action without any intermediate processing. However, real robot

systems following a behavior-based approach actually use various memory elements to handle the control problem correctly and efficiently; for instance, Brooks' subsumption architecture [4] contains memory in the augmented finite state machines. Memory provides a medium for processing of local temporal information in Behavior-based systems. However, there has been little research about the effect of varying amounts of memory in behavior-based robotics.

In some robotic tasks perceptual information alone is insufficient to determine uniquely the appropriate action. It may be necessary to integrate these partial perceptual cues with previous actions or perceptual information to decide proper actions. Many tasks, such as landmark-based navigation and sequential behaviors, require that an agent also be able to integrate perceptual information over time in order to determine the appropriate course of action [18, 19]. In evolutionary robotics, this ability is usually provided by evolving continuous-time recurrent neural networks which, for instance, can be evolved to generate a fixed sequence of ouputs in response to an external trigger occurring at varying intervals of time [19].

Evolutionary robotic approaches frequently use recurrent neural networks as controllers for complex behaviors [16, 9, 6], combined with genetic algorithms to determine the network structure or parameterization. Such networks make effective controllers, offering the possibility of creating many possible dynamical systems with a variety of kinds of attractor, but are hard to analyze them theoretically and to quantify the amount of memory a particular network provides to an agent.

Some evolutionary robotics experiments [15] have shown that pure reactive systems without memory can implement various primitive behaviors: for example, object avoidance and exploration behaviors, and box-pushing to a target signalled by an environmental marker need only a pure reactive controller with binary sensory inputs, and more complex combinations of such behaviors can be realized using stateless switchers to arbitrate between the primitives.

Rather than using neural networks, Lee [14] achieves these results by evolving a Boolean Logic Network using genetic programming (GP). Kauffman [11] modeled a genetic regulatory system using a Boolean Logic Network to represent the coupling between the various active or inactive genes. He observed that Boolean networks are a very powerful idealization with which we can consider a very broad class of continuous nonlinear systems. Switching systems are good idealizations of many nonlinear systems[11, 10].

In Lee's approach for robotics using Boolean Logic Networks [15, 14], each behavior is a pure reactive controller with no internal state. Its control structure is a combinational logic circuit, evolved using GP, acting on sensor values converted to binary by thresholding or comparison (the thresholds and comparisons are also determined by the GP evolution). Such an approach offers the possibility of adding memory in an easily quantified way.

A Boolean Logic Network with internal memory is equivalent to finite state machines[12]. Finite state machines have very useful properties for describing aggregations and sequences of behaviors [1]: they can represent the interaction of behaviors, the behaviors active at any given time and the transitions between

them. They have been used to accomplish various high-level goals with complex behavioral control systems consisting of a set of primitive behaviors [2, 7]. Finite state automata provide a good mechanism to express the relationships between various behavior sets and have been widely used within robotics to express control systems [1]. Brooks [4] also used a variation of finite state automata—augmented finite state machines—to express behaviors within his subsumption architecture.

Bakker proposed a means for counting the number of states required to perform a particular task in an environment[3]. Such state counts extracted from finite state machines can be a measure of the complexity of agents and environments. They applied their methodology to a fixed simple environment such as the Woods7 environment [17] instead of real robotic environments.

Wilson [17] used a zeroth-level classifier system (ZCS) for his animat experiments. The original formulation of ZCS has no memory mechanisms, because the input-output mappings from ZCS are purely reactive, but Wilson suggested how internal temporary memory registers could be added. Adding an internal memory register consisting of a few binary bits can increase the number of possible actions in the system.

Following Wilson's proposal, adding one-bit and two-bit memory registers to ZCS was tested in woods environments by Cliff and Ross [5]. They insisted ZCS manipulate and exploit internal states appropriately and efficiently in non-Markivian environments. Their experiments used simplified models and were quite different from real robotic experiments, but they showed the potential and importance of internal states for robot behaviors.

Similar to the ZCS experiments, Lanzi has shown that internal memory is effective with adaptive agents and reinforcement learning, when the agent's sensors provide only partial information about the surrounding environment and the situations require different actions [13].

## 2 Methods

To investigate the consequences of adding memory to a purely reactive controller, we have run several experiments evolving controllers for tasks such as obstacle avoidance, wall following, exploration, and box pushing using a simulated agent and environment. Our robot model is a Khepera-style robot, with binary sensors in front to detect any wall or obstacle. Those sensors are modeled as infrared sensors without any explicit acceptance cone[1]. The sensor values are binary, derived by thresholding the distance provided by the sensor, and sensor readings just say if any object is near robot within some limit in the direction of the sensor. These binary sensors do not give any quantitative value of how close to any obstacle the robot is. However, we allowed 3 bits, eight possible values, for each wheel action.

[1] The sensor detects the intersection of an object with a fixed length ray projected along the sensor's pointing direction and computes the distance to the nearest such intersection along the ray.

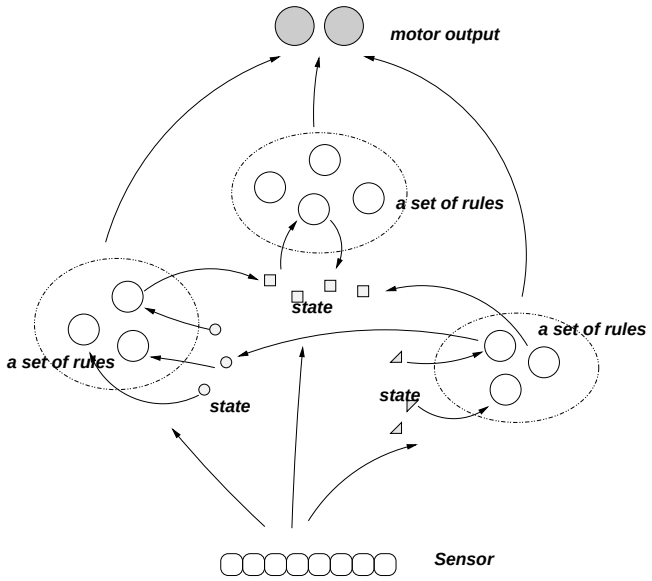


Figure 1: Diagram of a set of state rules

As mentioned above, we represent the fixed-memory controller as a collection of rules conditioned on the sensory state and on the controller's internal state. The rules thus implement the transition function of a finite state machine, and are similar to a classifier system [8]; they provide an economical way of representing the controller for evolution using a genetic algorithm. Our controller structure originates from finite state machines and the schemata expression of genes is a multivalued string instead of binary string. This scheme reduces the length of genes, thus increasing the convergence speed. In the experiments, the chromosome representation follows an integer encoding for output action and state number. When we have $n$ sensors, $m$ motor actions and $s$ states, the length of the chromosome for the direct representation of a finite state machine will be $s(m+1)2^n$, assuming all the sensors are binary-valued. For example, when we need many sensors and several states, the gene encoding size is exponentially increasing.

Using a set of rules, where each rule has a state information and a sensor representation of short, fixed, length as mentioned above, $k$ rules in a set, $n$ sensors, $m$ motor actions, and $s$ states results in a genome of length $k(n+s+m)$, which is linearly proportional to the number of sensors.

Genetic operators are used for a set of rules encoded by bit strings. The basic structure can be seen as a rule-based system where preconditions for a set of rules are examined to see their applicability, when given the current situation. The preconditions have fixed-length bit encoding with values of 0, 1, #(don't care). The action part of the rule is also a fixed-length encoding with the values of motor action. Unlike a classifier system, the evolutionary computation with

a set of rules used here does not use any reinforcement learning: the genetic algorithm creates new rules with genetic operators and a set of rules is tested for a given environment by evaluating the results of the chosen actions.

A set of rules including states is defined as a machine $M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$ where $q_0$ is an initial state, $Q$ is a finite set of states, $\Sigma$ is an input value, $\Delta$ is the multi-valued output value, $\delta$ is a state transition function from $Q \times \Sigma$ to $Q$, and $\lambda$ is a mapping from $Q \times \Sigma$ to $\Delta$, $\lambda(q, A)$ will have one of the output set $\Delta$, where $q, A$ is an element of $Q, \Sigma$, respectively. Each rule $R_i$ in machine $M$ is defined as

$$\begin{aligned} R_i(M) &= (A_i, (q_i, q_i^*), D_i) \\ &= ((\sigma_{i1}, \sigma_{i2}, \sigma_{i3}, ..., \sigma_{in}), \\ &\quad (q_i, q_i^*), (d_{i1}, d_{i2}, d_{i3}, ..., d_{im})) \end{aligned}$$

where $A_i$ is an input string or sensor value, $D_i \in \Delta$ is a motor output string, $q_i$ is the current state before rule activation, $q_i^*$ is the next state after rule activation.

In the rule structure, we can divide a set of state rules into several classes depending on the controller state $q_i$. We assume each rule must necessarily match the current state to be activated. State transition information can then be generated from the best rule to match the current sensor readings. As shown in Figure 1, the state information can be seen as a medium to activate a set of rules, switching the controller between different sensorimotor couplings. When the current state, e.g. $q_1$, is set, it can activate only a set of rules whose preconditions have state $q_1$ and the best rule among them will decide the next state $q_2$, which will activate another set of rules.

# 3 Experiments

We used a simulation program for robot with Khepera-like kinematics. At each time step, the robot reads a sensor value from the modeled sensors and the control action determined by the chosen rule is applied to the motors. We tested four different tasks—obstacle avoidance, wall following, exploration, and box pushing—with two sensor configurations: two infrared sensors with angle $45°, 135°$, or four infrared sensors with angle $45°, 80°, 100°, 135°$.

Each evolutionary run proceeds thus: $N$ different initial positions for the robot are randomly selected in the arena in Figure 2, to provide robustness in the evolved controller; evolutionary computation chooses $K$ positions randomly without replacement from the $N$ initial positions at each generation and evaluates the new control strategies at those positions and uses the fitnesses thereby computed for selection of parents in the genetic algorithm; every 10 generations controllers are re-evaluated at all $N$ positions. This $N$-$K$ sample method reduces the computation time, but also it has the effect of running over all initial positions regularly in the evolutionary run. We used $K = 5$ and $N = 10$ in our experiments. For significance statistics, we run 10 tests for each configuration of controller structures for each of four behaviors, respectively.

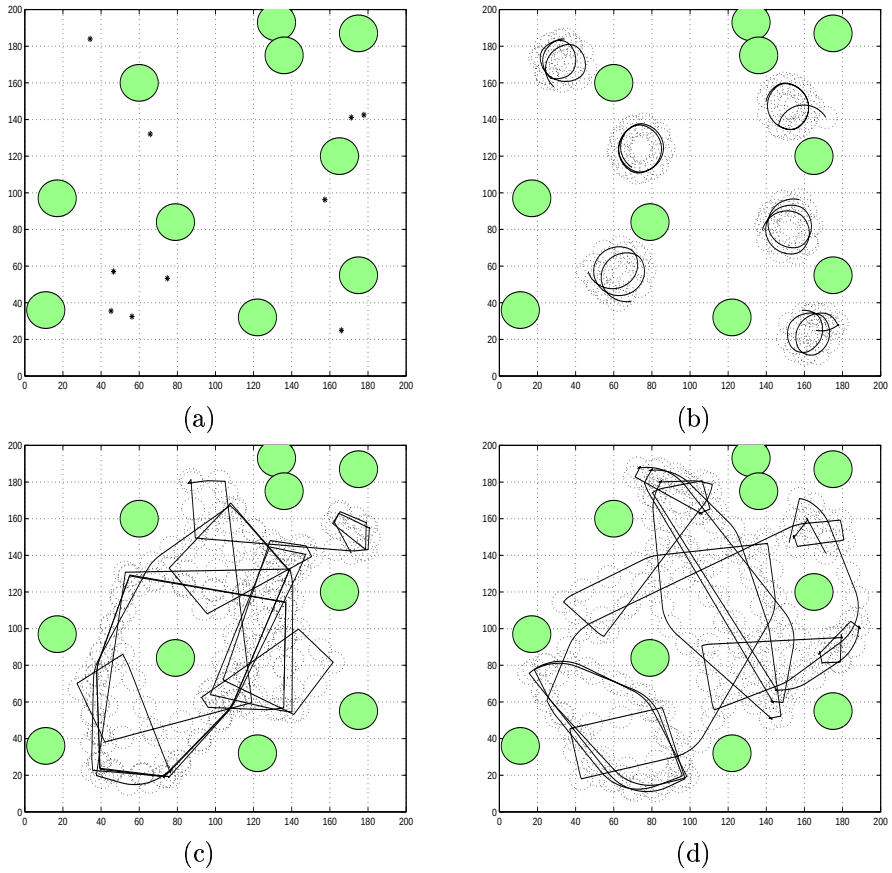


Figure 2: Obstacle avoidance behavior (a) arena with initial positions (b) reactive controller with 2 sensors (c) reactive controller with 4 sensors (d) 2 state machine controller with 4 sensors

### 3.1 Obstacle Avoidance

For obstacle avoidance behavior, we put circular boxes as obstacles randomly in the arena as shown in Figure 2. The fitness function $F_O$ is defined as a penalty function as follows:

$$\begin{aligned} F_O &= P - \Sigma_{t=1}^{max}(\alpha(M_L(t) + M_R(t)) \\ &\quad +\beta|M_L(t) - M_R(t)|/(2 * L) + \gamma C(t)) \end{aligned}$$

where $P$ is the base of penalty, $L$ is the robot wheelbase, $M_L(t)(M_R(t)$ the left(right) wheel motor output at time $t$, $C(t)$ is a collision detection function, and $\alpha = 0.1$, $\beta = 0.5$, $\gamma = 5.0$ are scaling coefficients for each term. If a robot collides with any obstacle, $C(t)$ will be set to one from that time onwards; otherwise it is zero. This collision factor encourages the controller to escape

obstacles without collision. Usually $\gamma$ is set to a high value, and when the robot experiences a collision the controller will have a very low fitness. The fitness calculates the difference between forward speed and rotation speed, to encourage motion as straight as possible. This experiment comprised controller evaluations lasting 2000 time steps, a population of 100 controllers and 4000 generations of the genetic algorithm.

For this task, memory-based controllers give a little better performance than reactive, but not significantly so. The number of sensors are more important as shown in Figure 2. When a robot has only two sensors, it cannot move straight forward due to collision with obstacles. It just staggers around a local area.

### 3.2 Wall Following

To evolve wall following behavior, we assumed there are many "energy tanks" in the arena the robot is exploring. If the robot finds one of these energy tanks, it acquires energy and can explore more area before its energy is exhausted. We assumed the robot loses its energy when moving. By placing the energy tanks near the walls of the environment and rewarding the agent for visiting them, we encourage the evolution of wall-following behavior: the fitness can be roughly calculated as the number of visited energy tanks and the energy level the robot has.

Thus, a fitness function $F_W$ is a penalty function defined as follows:

$$\begin{aligned} F_W &= P - (1 - 0.7C)\alpha\Sigma_i^N V(i) \\ &\quad -(1-C)(\beta E - \gamma T) \\ V(i) &= \begin{cases} 1 & \text{if energy tank } i \text{ is visited} \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

where $P$ is the base of the penalty, $C$ is a collision detection flag, $N$ is the number of energy tanks in the environment, $E$ is the remaining energy after exploration and $T$ is the number of time steps it takes the robot to visit all energy tanks. We assume energy tanks are located at divisions around the wall and obstacles as shown in Figure 3 on a 20cm by 20cm grid (the whole arena was 200cm by 200cm).

If it collides during exploration, robot movement stops and calculates the fitness function $F_W$ with $C = 1$, otherwise $C = 0$. Energy $E$ is changed every time step, by losing a small proportion of energy with initial energy setting.

In this experiment each controller generated by the genetic algorithm was evaluated for 2000 time steps; the population size was 100 and the evolution ran for 4000 generations.

In this task, a 2-state machine is significantly better than the pure reactive controller—see Figure 3(d), where we run 10 tests for each configuration and the error bar shows 95 % confidence interval. In some initial positions, the reactive controller cannot find the wall and it circles in the free space (see Figure 3(b)). The memory-based system is able to distinguish whether the robot has found a wall and to switch behavior when this occurs, as in Figure 3(c). The stateless

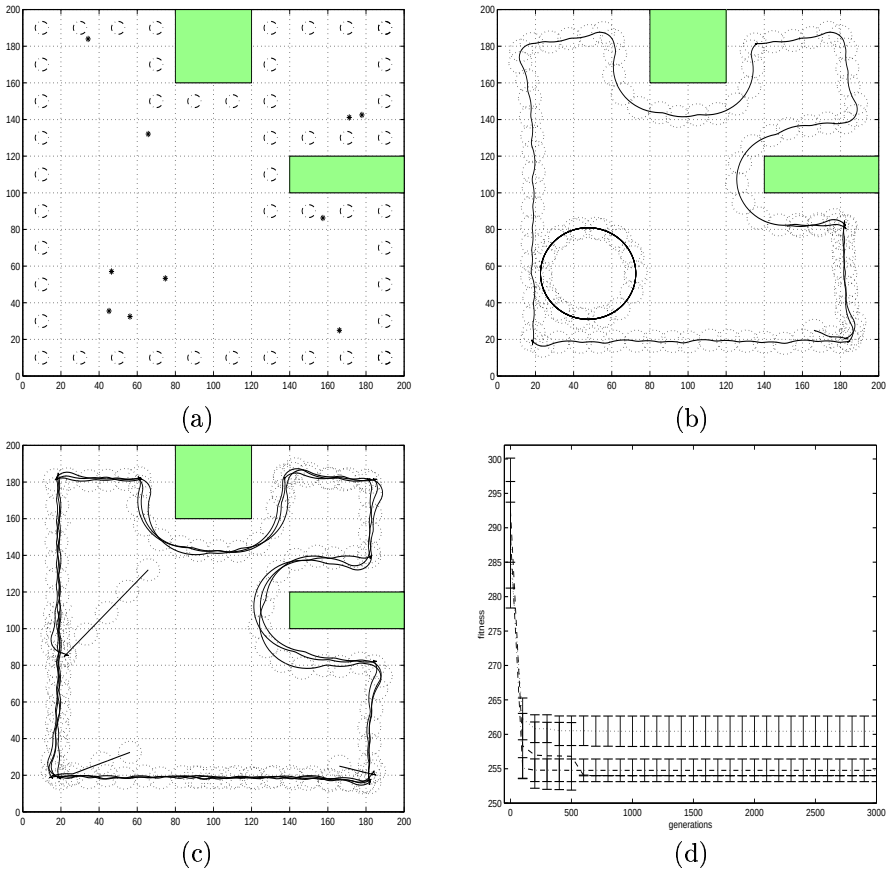


Figure 3: Wall following behavior (a) arena with initial positions and energy tanks (b) reactive controller (c) 2 state machine controller (d) 10 runs test

controller adopts an appropriately curved path to follow a wall, but in free space such a path results in circling motions. With the 2-state controller, if the sensors read clear, the robot moves straight forward until it finds a wall; after detecting a wall the robot switches to a curved path strategy. For this task, the number of sensors does not improve the behavior, and the robot with 4 sensors shows similar movements to that with 2 sensors.

## 3.3 Exploration

For exploration behavior, we used the same environment as that in the test of obstacle avoidance behavior as shown in Figure 4. For this behavior, we can apply the above energy distribution idea over the whole environment for the fitness function, but we used a different calculation method. The fitness

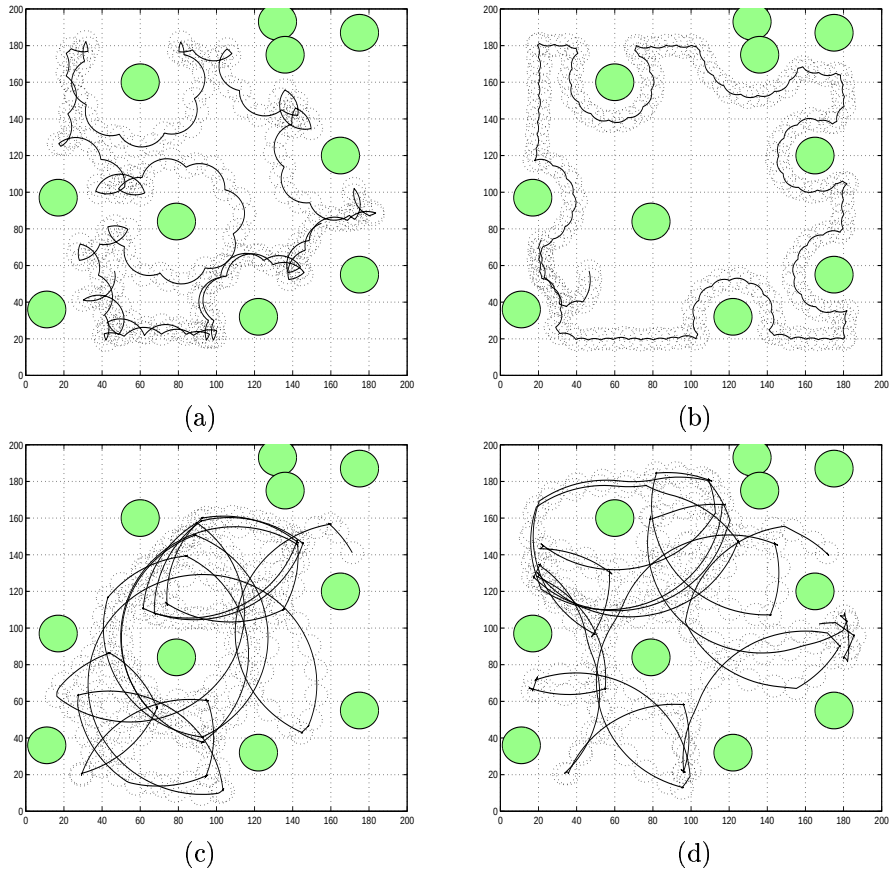


Figure 4: Exploration behavior (a) reactive controller with 2 sensors (b) 2 state machine controller with 2 sensors (c) reactive controller with 4 sensors (d) 2 state machine controller with 4 sensors

function is defined as follows:

$$F_E = -\Sigma_i^N p(i) \log p(i)$$

where $N$ is the number of small divisions, $p(i)$ is the probability that the robot has stayed in the $i$-th section and can be calculated dividing the number of time steps the robot has moved in the $i$-th area by the total time steps the robot runs.

Using this entropy measurement rewards the robot for visiting all areas of the environment uniformly, while the energy tank mechanism encourages the robot to visit the whole environment. As before, population size 100 with 4000 generations with 2000 time steps for each controller were used in each experiment.

There was no significant difference between reactive and memory-based controller in terms of fitness, but the state machine was a little better than a reactive controller on inspection of its behavior—see Figure 4. When a robot has two sensors, it shows wall following behavior. 3-state machine and 4-state machine behavior was similar to 2-state machine and in many cases 3-state and 4-state machines evolved into a 2-state machine controller. It appears that the 2-state machine is most desirable in this behavior. The 2-state machine controller shows a very delicate wall following behavior in Figure 4(b). Increasing the number of sensors allows the robot to move fast around the environment, because it reduces the possibility of collision by seeing the environment well.

### 3.4 Box Pushing

We assume a circular box is located in the arena and a robot moves the box into a target position indicated by a light. The robot uses its obstacle sensors to detect the circular box and ambient light sensors to detect the light. We simplified the light sensor value to binary values to say whether the light is close to the left or right sensor, or the light is positioned behind the robot. Control structures with a different number of memories were tested to see if the memory causes differences in the robot's performance. The fitness function $F_B$ is defined as a penalty function as follows:

$$F_B = \Sigma_{t=1}^{max}(\alpha W(t) + \beta D(t) + \gamma C(t))$$

where $W(t)$ is the distance between robot and box, $D(t)$ is the distance between box and light, and $C(t)$ is the collision detection function. In the experiment, we set $\alpha = 0.1$, $\beta = 1.0$, and $\gamma = 50.0$.

It was hard to evolve box pushing behavior with random initial positions—in a real application it would be combined with exploration behavior. 1500 time steps for each random gene controller were used with initial robot positions around the circular box. Robot controllers with a different number of sensors have show similar fitness values and behaviors. On varying the sensor pointing angles, $60°, 120°$, the memory-based controller was better than reactive

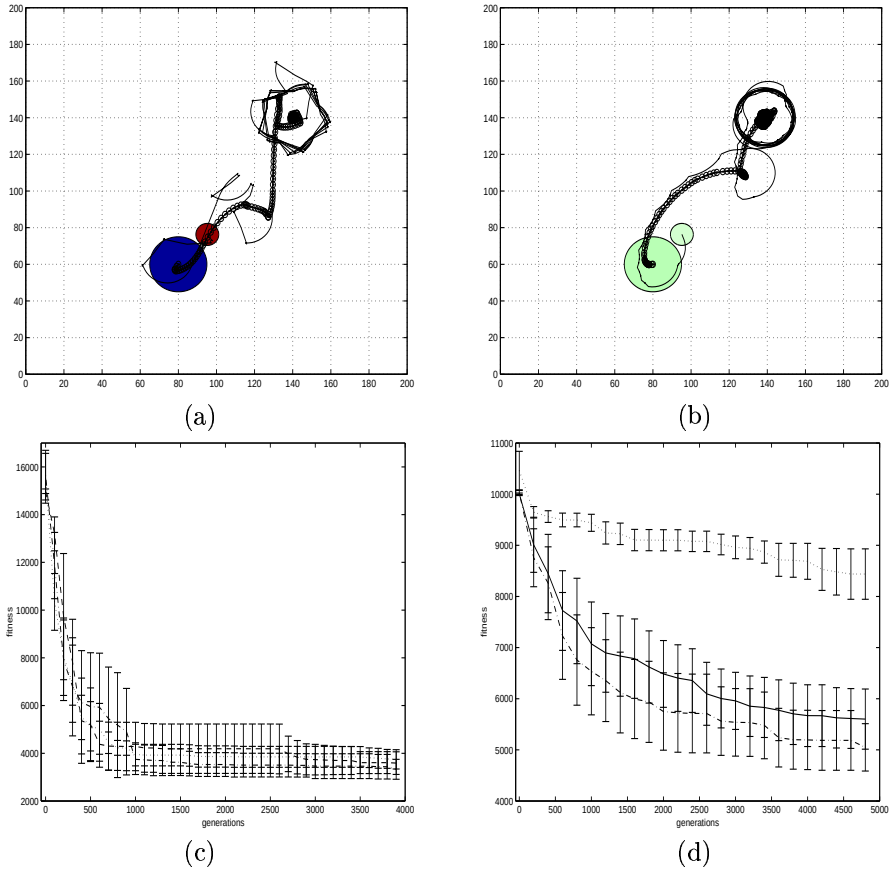


Figure 5: Box pushing behavior (a) reactive controller with 2 sensors (b) 2 state machine controller with 2 sensors (c) 10 runs test with angle 45 (d) 10 run test with angle 30 (dotted : reactive, dashed-dot : 2 states, solid : 3 states

controller as shown in Figure 5(d), where we run 10 tests for each configuration and the error bar shows 95 % confidence interval. The fitnesses of 2-state, 3-state, and 4-state machines are not significantly different.

# 4 Conclusion

In reactive systems, mobile robots work efficiently through a tight state-free coupling between perception and action. We investigated the effect of allowing the controller small amounts of memory in four different behaviors, obstacle avoidance, exploration, wall following, box pushing, and corridor following. In simple behaviors such as obstacle avoidance and exploration, a reactive system without memory elements is sufficient to control a mobile robot. The number of sensors and each sensor position have more influence on performance of such behaviors than the amount of state available.

In some behaviors, such as wall following, box pushing and corridor following, the memory element plays an important role by allowing the controller to specialize its behavior depending on context—for instance, free-space versus close to a wall. The amount of memory available is not significant for these tasks as tested here, because the tasks themselves only require a couple of states to be distinguished.

Future work will look at the effect of memory on behavior and performance when the evolutionary system is able to choose the sensor thresholds as in Lee's work [14] (here the thresholds were determined by the experimenter) and on the effect of incorporating memory into the stateless arbitrators Lee used to combine his primitive behaviors (for instance to combine exploration and box-pushing).

# References


[1] R.C. Arkin. *Behavior-based Robotics*. MIT Press, Cambridge, MA, 1998.

[2] R.C. Arkin and D. MacKenzie. Temporal coordination of perceptual algorithms for mobile robot navigation. *IEEE Transactions on Robotics and Automation*, Vol. 10, No.3:276–286, June 1994.

[3] B. Bakker and M. de Jong. The epsilon state count. In *From Animals to Animats 6: Proceedings of the Sixth International Conference on Simulation of Adaptive Behaviour*, pages 51–60. MIT Press, 2000.

[4] R. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Autonomation*, Vol. RA-2, No.1:14–23, 1986.

[5] D. Cliff and S. Ross. Adding temporary memory to zcs. *Adaptive Behavior*, 3(2):101–150, 1995.

[6] D. Floreano and F. Mondada. Evolution of homing navigation in a real robot. *IEEE Transactions on Systems, Man and Cybernetics*, 26(3):396–407, 1996.

[7] E. Gat and G. Dorais. Robot navigation by conditional sequencing. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1293–1299, 1994.

[8] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison Wesly, Reading, MA, 1989.

[9] I. Harvey, P. Husbands, and D. Cliff. Issues in evolutionary robotics. In *From Animals to Animats 2: Proceedings of the Second International Conference on Simulation of Adaptive Behaviour*, pages 364–373. MIT Press, 1992.

[10] J.J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79:2554–2558, 1982.

[11] Stuart A. Kauffman. *The Orgins of Order : self-organization and selection in evolution.* Oxford University Press, New York, 1993.

[12] Zvi Kohavi. *Switching and Finite Automata Theory.* McGraw-Hill, New York, London, 1970.

[13] P.L. Lanzi. Adaptive agents with reinforcement learning and internal memory. In *From Animals to Animats 6: Proceedings of the Sixth International Conference on Simulation of Adaptive Behaviour*, pages 333–342. MIT Press, 2000.

[14] W-P. Lee, J. Hallam, and H.H. Lund. Applying genetic programming to evolve behavior primitives and arbitrators for mobile robots. In *Proceedings of IEEE International Conference on Evolutionary Computation*, Indianapolis, USA, 1997.

[15] Wei-Po Lee. *Applying Genetic Programming to Evolve Behavior Primitives and Arbitrators for Mobile Robots.* Ph. D. dissertation, University of Edinburgh, 1998.

[16] J.A. Meyer, P. Husbands, and I. Harvey. Evolutionary robotics: A survey of applications and problems. In *First European Workshop on Evolutionary Robotics*, Paris, France, 1998. Springer.

[17] S. W. Wilson. Zcs: A zeroth level classifier system. *Evolutionary Computation*, 2(1):1–18, 1994.

[18] B. Yamauchi and R. Beer. Integrating reactive, sequential, and learning behavior using dynamical neural networks. In *From Animals to Animats 3: Proceedings of the Third International Conference on Simulation of Adaptive Behaviour*, pages 382–391. MIT Press, 1994.

[19] B. Yamauchi and R. Beer. Sequential behavior and learning in evolved dynamical neural networks. *Adaptive Behavior*, 2(3):219–246, 1994.