Generating Adaptive Behavior using Function Regression within Genetic Programming and a Real Robot

Wolfgang Banzhaf, Peter Nordin, and Markus Olmer

Department of Computer Science, Dortmund University,

Joseph-vonFraunhofer-Str. 20, 44227 Dortmund, GERMANY

Phone: +49-231-9700-953956/977 Fax: +49-231-9700-959

banzhaf,nordin,olmer@LS11.informatik.uni-dortmund.de

Abstract

We discuss the generation of adaptive behavior for an autonomous robot within the framework of a special kind of function regression used in compiling Genetic Programming (GP). The control strategy for the robot is derived, using an evolutionary algorithm, from a continuous improvement of machine language programs which are varied and selected against each other. We give an overview of our recent work on several fundamental behaviors like obstacle avoidance and object following adapted from programs that were originally random sequences of commands. It is argued that the method is generally applicable where there is a need for quick adaptation within real-time problem domains.

Keywords: Autonomous robots, Evolutionary computation

1 Introduction

The study of adaptive behavior has two main thrusts. One is the description and analysis of animal behavior, the other is the generation of behavior for artificially created autonomous agents. One prominent class of autonomous agents are robots acting in the real world. The aim of generating behavior for robots is to make them more and more useful companions of human beings. In this contribution we shall report on recent work we have done in the realm of robotics for generating behavior. We shall be concentrating here on very fundamental behavior for a mobile robot, like obstacle avoidance or moving around, for providing clearer insights into the central ideas of the approach.

Robots do have to act in the real world. That means they have to execute actions, based on the internal representation of the world that they have acquired or have been equipped with, and based on the sensory information which they perceive by various sensors about the status of the environment. In other words, robots need means to process incoming information and to control actuators for generating effects in the environment. Although earlier on this has been achieved with impressive capabilities with simple analog electronic devices [2, 15], nowadays most robots are equipped with at least one central processing unit, be it an off-the-shelf processor including its setting, or be it one or more special embedded processor(s), like a DSP, for fast reaction.

In this way, the need for wiring and connecting analog electronic devices has now been substituted by the need for programming the processor devices in robots. From a behavioral point of view, programs are abstract descriptions of behavior, in most cases behavior for a processor shifting and processing data within memory. In certain special cases, namely when memory values are read out into physical observables, causing e.g. electric motors to turn, they code for behavior in the real world. We are interested here in this special case that underlies modern robotics.

Our approach is to look, at a very fundamental level, into the adaptivity of computer programs. If computer programs can, under the condition of being executed, be interpreted as behavior, adaptive behavior could be generated simply by allowing computer programs to adapt. One method to do this is called Genetic Programming (GP) [17]. It has roots back into the fifties [9, 6] but has recently become an object of intense study in Computer Science [18]. GP tries to apply insights from natural evolution which had been gleaned already earlier for the purpose of optimization in technical systems [28, 31, 14, 10, 8]. By using a population of programs which all behave slightly different, plus differential selection according to a behavioral fitness criterion, programs can be bread that are better adapted to certain conditions than others. Note that this adaptivity happens in the very heart of computing: in programs. We do not employ neural nets [5, 12, 7] or optimize parameters specifying the strength of presence of certain behavior [1, 13]; we try to evolve those behaviors from scratch.

Similar experiments have been done earlier [17, 30, 11] or proposed [3] but have so far been restricted to simulated environments. We tried to port these concepts into the harsh environment of true reality which has a lot of difficulties never encountered in simulations [4, 16, 19]. In previous work we have reported on different series of experiments with our approach [24, 25, 26, 27]. The present contribution tries to summarize the status of our work and the progress we made so far.

2 Implementation

2.1 The robot

For our studies we used a standard autonomous miniature robot, the Swiss mobile robot platform Khepera [20]. It is equipped with eight infrared proximity sensors. The mobile robot has a circular shape, a diameter of 6 cm and a height of 5 cm. It possesses two motors and on-board power supply. The motors can be independently controlled by a PID controller. The eight infrared sensors are distributed around the robot in a circular pattern. They emit infrared light, receive the reflected light and measure distances in a short range of 2 to 5 cm. The robot is also equipped with a Motorola 68331 micro-controller which can be connected to a SUN workstation via serial cable. A schematic figure of the robot is shown in Figure 1.

The robot moves around in the environment of Figure 2.

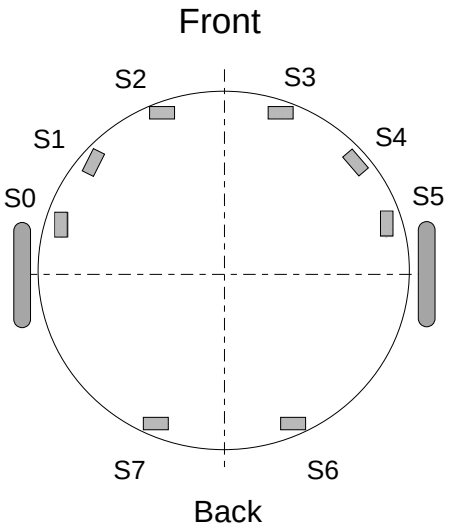

Figure 1: Schematic view of the Khepera robot.

Figure 2: The Training Environment. Movable objects and the Khepera in inside the boundaries.

2.2 The GP system

There are some reasons to argue in general for use of Genetic Programming for the purpose of generating adaptive behavior in processor-controlled systems like the Khepera robot:

- With a general purpose programming language, any sort of behavior can be produced.
- GP generates purely symbolic output. This is to be seen in contrast to what neural nets produce: a list of parameter values.
- The definition of goals is easily achieved by deciding one fitness function only.
- GP has a built-in tendency to generalize from presented situations. That has to do with the fact that GP evolves algorithms or, in other words, recipes for behavior that must, by definition, be applicable to more than one situation.

Our specific implementation is a GP system which manipulates machine code directly [21]. The system uses linear genomes composed of variable length strings of 32 bit instructions for a register machine. Each node in the genome is an instruction for the processor. It performs arithmetic or logic operations on a small set of registers. Each instruction might also include a small integer constant of maximal 13 bits. The actual format of the 32 bits corresponds to the machine code format of a RISC processor. The genetic operators act on binary code directly. Crossover, for instance, occurs between 32 bit instructions, thus changing the order and number of instructions used in offspring programs. Mutation is allowed to flip bits within an instruction. The mutation operator, however, has to ensure that only those instructions are called which are in the function set and that the defined ranges of registers and constants are obeyed by a variant genome. For a more thorough description of this system the reader is referred to [22, 23].

The function set used in our experiments is comprised of the following machine code instructions: Arithmetic operations ADD, SUB and MUL, register shift operations SLL and SLR and logic operations AND, OR and XOR. All of these instructions operate on 32-bit registers.

In addition to the general advantages of GP mentioned above we see some advantages in applying a low-level implementation of the paradigm such as the one used by us:

- This approach is efficient: It is up to 2000 times faster than an interpreting GP system and has constant memory requirements with a very small kernel of 32 kB.
- A minimum of a priori knowledge is necessary because the only task specific knowledge that has to be added is the fitness equation, which is without procedural or representational knowledge. The representation also shows a minimum of prejudice because behavior is coded at the lowest level of the processor, its binary code.

3 Three Approaches to Adaptive Behavior of Real Robots

Before we discuss in more detail the behavioral results of our experiments we present the three steps taken so far in increasing complexity. Because the general set-up will always be the same, we shall report it here for all of the tasks discussed below.

3.1 Function regression and behavior

A particular method for achieving behavior is to consider it as the values returned from a function[1] that get executed physically. The input to this function would usually be composed of sensory values and internal state variables of the behaving system. The function would try to match in the best way possible the reaction of the environment to the previous behavioral commands, thus for instance, building a world model and trying to behave appropriately.

As it turns out, Genetic programming is ideal for symbolic function regression, and most GP applications could be reformulated as variants of function regression. That is because GP handles algorithms,

[1] Function and program are in this approach basically identical

Degree of difficulty	single task	single task	multiple task
Memory ?	without memory	with memory	without memory
Description	Avoiding obstacles	Avoiding obstacles	Avoiding obstacles seeking objects following objects following walls hiding in the dark selecting an action

Table 1: Different tasks studied.

and tries to optimally approximate numerical data with symbolic or mathematical functions. In the robot control application a function could approximate the reaction of the robot to sensor values returning values that are interpreted as motor commands thus leading to behavior of the robot.

A very general form of this control function could be:

$$\{b_1, ..., b_i, ..., b_I\} = \mathbf{f}(s_1, ..., s_j, ..., s_J, z_1, ..., z_k, ..., z_K) \tag{1}$$

with b_i behavioral variables, s_j sensor values and z_k internal state variables.

In our case, the low-level commands of the binary machine code can be disassembled into a symbolic form and would read, for instance:

```
a=s3 + 4;
d=s2 >> s1;
b=s1 - d;
d=s2 + 2;
c=d >> 1;
b=d - d;
...
e=c >> b;
motor2=a | e;
c=d | 7;
motor1=c * 9;
c=e & e;
```

This code is part of an actual program evolved by the system where $s_0 - s_7$ are the input sensor values, $a - e$ are registers for temporary storage of values and $motor1, motor2$ are the resulting speed values that are sent to the motors[2]. We can identify input processing, internal processing and output computation.

A population of programs trying to control the robot are run within the evolutionary algorithm in tournament selection mode. We use small population sizes, typically less than 50 individual programs. Each individual program does its manipulation independent of the others and thus stands for an individual behavior of the robot if invoked to control the motors.

Table 1 reports the different tasks we have been studying so far with the Khepera robot.

3.2 Memory-less GP system for single task

The goal of the controlling GP system which has no memory except its population of programs is to evolve obstacle avoiding behavior. As all the systems reported here, it operates real-time and aims at obstacle avoiding behavior based on data from noisy sensor devices. The sensorial data come from six of its eight infrared proximity sensors. The problem domain is already well known, and different methods have been discussed in the literature for achieving this behavior [2, 29, 19, 32]. Figure 3 shows a diagram of the system.

[2] Besides arithmetic operators we recognize << and >>, the logical shift operations.

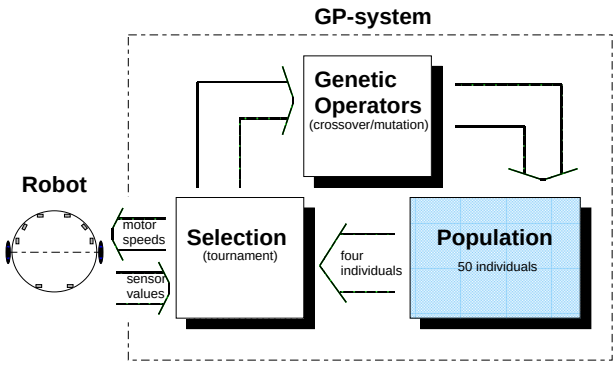

Figure 3: Schematic View of the Control System.

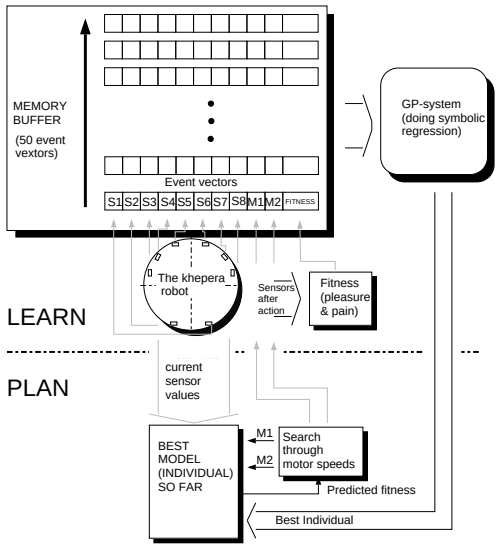

Figure 4: Schematic view of the memory based control architecture. Specific are: (a) The memory buffer storing event vectors representing events in the past, (b) a memory buffer storing event vectors, (c) a search module that tries to find the best action given the currently best world model, (d) the currently best induced individual model

3.3 Memory-based GP system single task

So far, all reactions belong to the rather simple stimulus-response type. Although it was possible to achieve obstacle avoiding behavior the robot gave the impression of reacting like an insect, wandering around, then suddenly changing direction due to an obstacle detected. Clearly, a system with memory should behave much more sophisticated, so we decided to do a second series of experiments allowing the system to make use of memory. A more serious limitation of the non-memory system it that its learning speed is, to a large extent, determined by the dynamic response time of the environment not by the speed of the computer hardware or the learning algorithm. This limitation is removed when using historical events: The system can run at full speed contemplating history, not only the present event.

The memory-based control architecture now consists of two separate processes. One process is communicating with sensors and motors as well as storing events into the memory buffer. The other process is constantly trying to learn and induce a model of the world consistent with the entries in the memory buffer.

We call the former process the *planning* process, because it is involved in deciding what action to perform given a certain model of the world. The latter process is called the *learning* process, because it consists of trying to derive a model (in the form of a function) from memory data.

Figure 4 system gives a schematic illustration of the architecture of the control system.

The result of this method shows that the GP learning algorithm is accelerated by a factor of 1500 and the learning of behavior is consequently speeded up by 40 times allowing it to learn obstacle avoiding

bahaviour in a few minutes [25].

3.4 Memory-less GP system for multiple tasks

Given our earlier experiments with a memory-less system for a single task, we asked ourselves, whether it would be possible to extend the approach by requiring more than one behavior to be learned independently.

The motivation for this question was that not all behavioral problems are as easy as obstacle avoidance. Solutions to more complex problems would be hard to evolve directly. Hence, a division of the task into smaller sub-tasks was seen as a possible way out. In this setting, the GP system first learns the sub-tasks and then evolves a higher-level action selection strategy for deciding which of the evolved lower-level algorithms should be in control.

Thus, the goal of the GP system is to evolve several primitive behaviors plus an action selection behavior within the same run. The individuals are separated into different isolated populations which use various parts of the full sensorial data from the robot's sensors (8 ambient light, 8 reflected light, two measured motor speed, two distance traveled) as input and produce two motor speeds as output. The action selection population also uses the fitness values of the action primitives computed over the last trials.

The five populations for the action primitives and the action selection module are:

GO AHEAD : the robot learns to move straight ahead at maximum speed.

AVOID OBSTACLE : the robot avoids obstacles at a learned maximum speed.

SEEK OBJECT : in a way the inverse of *AVOID OBSTACLE*, useful to create wall following behavior. Wall following would be an oscillating task switching between obstacle avoidance and object seeking.

HIDE IN THE DARK : the robot searches for a dark corner to hide in.

SELECT ACTION : this population contains functions which select one of the above action primitives.

When the robot starts learning, the system feeds the required data and sensor readings into the input register space of the action selection mechanism population. Four individuals are selected for the action selection tournament. Every chosen individual selects one of the four action primitives. Repeatedly required values are copied into the GP register space and the tournament for an action primitive starts. The winners replace the losers and the genetic operators are used. This is done for all selected individuals from the action selection population. Afterwards the GP system updates the selection population.

4 Behavior

Here we shall discuss various aspects of the evolved behavior. The general results show fast learning of behavior and good generalization properties but as it turned out, in the early stages of the experiments, the system was very inventive in finding short-cuts and unexpected, often amusing, ways to fulfill the fitness criteria without actually doing what we intended it to do.

4.1 Learning to avoid obstacles

The system had to associate high sensory values with closeness to an object or obstacle or, with a high probability to suffer a collision. We therefore used the sum of sensory values as its fitness function. In doing so, however, a simple stand-still of motors wherever the robot is located (except near to objects) would fare perfect. Hence we had to add a second term to the fitness that rewarded movement in a straight direction. Thus, the fitness for the first task was:

$$f = \sum s_i - (m_1 + m_2 - \mid m_1 - m_2 \mid) \tag{2}$$

We can already see here, that the system had to use positive as well as negative measures. We called them, due to their unspecificity, pleasure and pain, alluding to elementary feelings of real organisms.

Note that, with this fitness function, the system has no clue where the obstacle is located. Approaching an obstacle with right sensors might cause the same negative fitness contribution as approaching it with left sensors. This fitness function could be compressed into the more general formula:

$$f = \sum_{i=1}^{8} s_i - \sum_{j=1}^{2} m_j \tag{3}$$

The difference is here that the robot gets no award for moving straight.

4.2 Learning to seek objects

Learning to seek objects cannot be just the opposite of the above, since that would mean movement would cause pain and the system would try to stop at the next wall.

Instead, the fitness function for seeking objects is using the front sensors only and encouraging movement towards an object[3]:

$$fitness = 4000 - (s_1 + s_2 + s_3 + s_4) \tag{4}$$

4.3 Learning to follow objects

Learning to follow objects was even more challenging, because it required moving objects with presumably unpredictable velocities and thus obstacle avoiding and object seeking behavior at the same time.

The fitness calculation for the object following task is based on the four sensors facing forward. In order to define the desired behavior we needed to give a function which attracted the robot to objects far away but repelled them if they are too close. The fitness function thus had an U-shape where the lowest pain is at a predefined distance from an object:

$$fitness = (s_1 + s_2 + s_3 + s_4 - 1000)^2 \tag{5}$$

The value of 1000 represents an ideal distance from an object. When the sum of the front facing sensor are 1000 then the agent perceives its "ideal feeling" and the fitness is zero.

4.4 Learning to follow walls

The next task for the robot was to follow the walls of the experimental environment. It also comprised a combination of obstacle avoiding and object seeking behavior. This time, we measured the number of times when the robot exits a narrow space (4 cm) between the robot and the wall. Here the important sensors were the side sensors:

$$fitness = (s_0 - 1000)^2 + (s_1 - 500)^2 + s_3^2 - (m_1 + m_2)^2 \tag{6}$$

The robots tries to keep the side sensors (at one side) at certain distances while moving forward and avoiding stimuli on the corresponding front sensor[4].

4.5 Learning to hide

The sensors of the robot do not only provide data on distance from obstacles (in distance mode) but also brightness data in the light condition mode. This information can be used for the robot to turn towards areas with slower brightness, such as shadowy or even dark places where it could then try to hide. The same fitness function as in equation 2 was used, just s_i now meant ambient light, not distance from an obstacle.

$$f = \sum s_i - (m_1 + m_2 - \mid m_1 - m_2 \mid) \tag{7}$$

[3] The constants used in the fitness functions reflect the fact that each of the sensors has a range of integers 0...1023

[4] In reality all fitness equations are tried out with different weights on the different equation components.

4.6 Action selection

A level higher in the hierarchy was the action selection module that also received the lower level (behavioral primitives') fitness as an input. As it turned out it was not as easy to evolve realizable selection strategies. The system was not able to discern the hierarchical levels until we introduced a separation of time-scales by allowing the higher level module to take into account an average fitness of lower level modules. Which other way could the system have to discern the hierarchical levels?

4.7 Robustness of the population approach

An interesting aspect of the approach used is its robustness against perturbations of the environment. At unpredictable moments we had to take the robot out of its environment and turn the cable which threatened to remove the robot from ground due to a one-directional spin. The robot was subsequently never put down at the same place, with no obvious effects on its performance. Even if we put it into a completely new environment it performed quite well.

5 Interpretation

In this section we want to interpret the behavior seen during numerous experiments over the last year. However, several questions about the underlying mechanisms of the behavior are unanswered. We do not know yet, for instance, how the individual programs behave during evolution. They are set in a competitive algorithm, but do they compete or cooperate?

5.1 Cooperation and competition between programs

The population of programs is in a delicate balance, since any one of the programs does get a small time-slice when it is allowed to control the robot. A successful program does not receive a larger time-slice, rather it will get exactly the same time to steer as any other, and maybe worse, program. Due to the nature of the environment, a time-delay of 500 ms is allowed before the evaluation of the action of a program takes place that ultimately leads to a fitness score for that particular program.

There are, in principle, two strategies for programs to survive in the population: (i) either it manages to fulfill the fitness criterion very well, even under bad initial conditions, providing e.g. for backing-up commands in case of a probable obstacle collision, (ii) or it tries to minimize the fitness of its competitors by steering the robot into a situation which will be very difficult to handle for any successor in the subsequent time-slices. We call the former a *cooperative* strategy in that it might usually leave the robot in a better situation than it encountered itself. Conversely, the latter strategy is a *competitive* strategy, as it tries to maximize penalty for successors and in this way achieves a differential fitness advantage.

5.2 Memory, emotion and childhood

It is interesting to note that within the memory-based approach, a sort of"childhood" has to be introduced which allows the system to gain good and bad experiences without forgetting them. Our first approach to managing the memory buffer when all 50 places had been filled was to simply shift out the oldest memories as the new entries came in. However, we soon realized that the system then forgot important early experiences. We found out that early mistakes made before a good strategy was found are crucial to remember in order to not evolve world models that permit the same mistakes to be done again. Hence we gave the robot a childhood – an initial period of quick learning whose memories were not so easily forgotten. The robot performs better if it has a harder time memorizing events later on when it it grows older – a mechanism all too familiar to humans as well. The childhood period also reduced the likelihood that the system displays a very special strategy: Only to perform those actions which effectively confirm its current (limited) world model.

Another important factor for successfully inducing an efficient world model is to have a *stimulating* childhood. It is important to have a wide set of experiences to draw conclusions from. Noise is therefore added to the behavior in the childhood to avoid stereotypic behavior very early in the first seconds of the system's execution. As long as experiences are too few to allow for a meaningful model of the world,

this the noise is needed to assure early experiences of sufficient diversity. This noise causes the robot to behave similar to a newborn, with uncontrolled arbitrary movements.
Later on, as basic knowledge in the form of a working world model has been gained, it is not so important to store every sort of experience, and the system should become more choosy about what to add to its memory. We speculate that a good measure to use for discrimination between important and unimportant events, thus paving the way for either storage or discard of the corresponding data, would be the intensity of feelings or, if you want, the degree of emotion, which is attached to a certain situation. If a situation is accompanied with much pain, for instance, this might well be worth remembering, as it opens up the possibility of the system to do function regression over a much wider range of parameters. If all experiences stored would be similar in terms of the accompanying feelings (encoded as the value of the fitness function), this would cause the robot to be unprepared for extreme situations which might suddenly realize, thus rendering him incapable of handling those situations appropriately.

5.3 Exploration and curiosity

One other interesting aspect of the existence of more than one program allowed to steer behavior is — mainly in the beginning — the strong impression of explorative behavior the system evokes within an observer. Although there is no such elementary behavior as exploration built into the system, this behavior emerges from the interaction of initially random programs which all get to steer the robot for some time-slices.

5.4 Character

The behavior of the robot is very different when memory is added to the system. Then the system displays a set of very "reasonable" behaviors. The robot regularly displays a clear strategy and travels in straight lines or smooth curves. Some of the behaviors evolved show an almost perfect solution to the current task and fitness function, see Figure 5. Because the robot performs an optimization based on its world model which is reaffirmed any time it is gaining experiences modelled within it, we have termed the different behaviors "characters".

In this contribution we tried to classify the behavior during evolution in our experiments. Some of the emerging intelligent strategies are illustrated and explained in Figure 5.

6 Conclusions

We tried to give a comprehensive overview of what has been achieved in our laboratory so far with using a machine language implementation of GP and a real robot platform. Though more quantitative data on the experiments can be found elsewhere, we hope that the potential of the method has become evident. For each of the behaviors we simpy had to change the fitness function.

In the future, we would like to further investigate the memory approach to behavior, the hierarchy of behavioral mechanisms, including action selection, as well as more low-level behavior like fleeing and hiding, which are also very fundamental in organisms.

Acknowledgments

This work has been supported by the Deutsche Forschungsgemeinschaft (DFG) under grant Ba 1042/5-1. Also, we would like to acknowledge support from the Ministerium für Wissenschaft und Forschung des Landes Nordrhein-Westfalen, under grant I-A-4-6037.I for buying a Khepera robot platform.

References

[1] A. Ram, G. Boone, R. Arkin, M. Pearce, Using Genetic Algorithms to Learn Reactive Control Parameters for Autonomous Robotic Navigation, *Adaptive Behavior* **2** (1994) 277 -305

[2] V. Braitenberg, *Vehicles*, MIT Press, Cambridge, MA, 1984

[3] R. Brooks, Artificial Life and Real Robots, in: *Proc. of the 1st European Conference on Artificial Life (ECAL-91)*, F.J. Varela, J. Bourgine (eds.), MIT Press, Cambridge, 1992

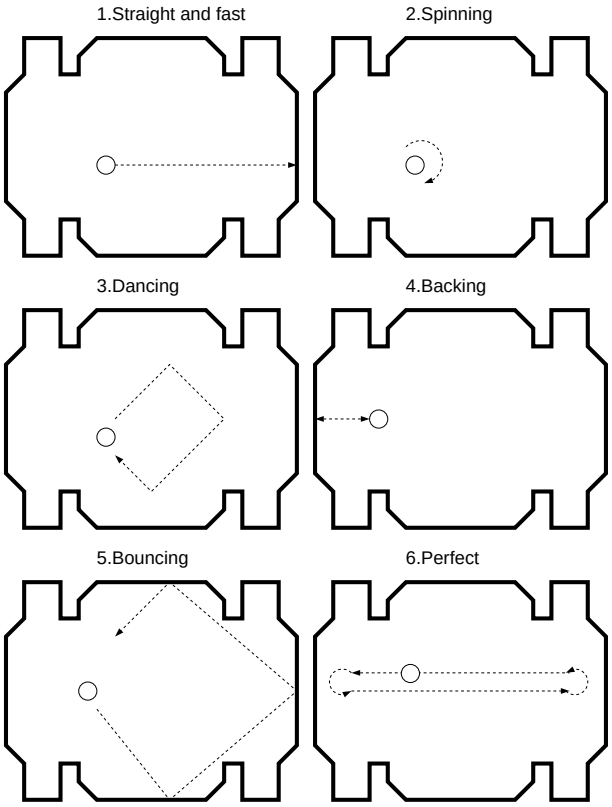

Figure 5: Different common strategies that evolves when obstacle avoidance was trained with the memory-based system: (1) *Straight and fast*, the robot heads into the nearest wall and tries to break through it with spinning wheels; (2) *Spinning*, the robot starts spinning around its own axis avoiding all obstacles; (3) *Dancing*, with the use of state information the robot navigates to the open space where it starts to move in an irregular more or less circular path avoiding obstacles; (4) *Backing–up*, the robot, while moving around, avoids abstacles by backing up; (5) *Bouncing*, the robot gradually turns away from an obstacle as it approaches it; (6) *Perfect*, the robot uses the large free space in the middle of the training environment to go straight and fast, until it senses an object when it quickly turns 180 degrees and continues going straight in the opposite direction.

[4] R. Brooks, Intelligence without Representation, *Artificial Intelligence* **47** (1991) 139 - 159

[5] D. Cliff, Computational Neuroethology: A Provisional Manifesto, in: *From Animals To Animats: Proceedings of the First International Conference on simulation of Adaptive Behavior*, J. Meyer and S. Wilson (eds.), MIT Press, Cambridge, MA, 1991

[6] N.L. Cramer, A Representation for Adaptive Generation of Simple Sequential Programs, In *Proc. of the 1st International Conference on Genetic Algorithms* , 183 - 187, 1985

[7] D. Floreano, F. Mondada, Automatic Creation of an Autonomous Agent: Genetic Evolution of a Neural-Network Driven Robot, in: *From Animals to Animats III: Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, D. Cliff, P. Husbands, J. Meyer and S.W. Wilson (eds.), MIT Press-Bradford Books, Cambridge, MA, 1994

[8] D. Fogel, *Evolutionary Computation*, IEEE Press, New York, 1995

[9] R.M., Friedberg, A Learning Machine - Part I, *IBM Journal of Research and Development*, IBM, **2(1)**, 2 - 11, 1958

[10] D. Goldberg, *Genetic Algorithms in Search Optimization & Learning*, Addison-Wesley, Reading, MA, 1989

[11] S. Handley, The Automatic Generation of Plans for a Mobile Robot via Genetic Programming with Automatically Defined Functions, in: *Advances in Genetic Programming*, K. Kinnear (ed.), MIT Press, Cambridge, MA, 1994

[12] I. Harvey, P. Husbands, D. Cliff, Issues in Evolutionary Robotics, in: *From Animals To Animats 2: Proceedings of the Second International Conference on simulation of Adaptive Behavior*, J. Meyer and S. Wilson (eds.), MIT Press, Cambridge, MA, 1993

[13] I. Harvey, P. Husbands, D. Cliff, Seeing the Light: Artificial Evolution, Real Vision, in: *From Animals To Animats3: Proceedings of the Third International Conference on simulation of Adaptive Behavior*, D. Cliff, P. Husband, J. Meyer and S. Wilson (eds.), MIT Press, Cambridge, MA, 1994

[14] J. H. Holland, *Adaption in Natural and Artificial Systems*, The University of Michigan Press, Ann Arbor, MI, 1975

[15] O. Holland, C. Melhuish, Demonstration of Grey Walter's Tortoises, *Abstracts book, Europ. Conf. on Artificial Life, 1995*, Granada, 1995

[16] N. Jakobi, P. Husband, I. Harvey, Noise and the Reality Gap: The Use of Simulation in Evolutionary Robotics, in: F. Moran, A. Moreno, J.J. Merelo, P. Chacon, *Advances in Artificial Life, Proc. of ECAL-95*, Springer, Berlin, 1995

[17] J.R. Koza, *Genetic Programming*, MIT Press, Cambridge, MA, 1992

[18] See the growing GP-bibliography, maintained by William Langdon and available via ftp://cs.ucl.ac.uk/genetic/biblio/README.html

[19] M.J. Mataric, Designing Emergent Behaviors: From Local Interactions to Collective Intelligence, in: *From Animals To Animats 2: Proceedings of the Second International Conference on simulation of Adaptive Behavior*, J. Meyer and S. Wilson (eds.), MIT Press, Cambridge, MA, 1993

[20] F. Mondada, E. Franzi, P. Ienne, Mobile Robot Miniaturization. in: *Proceedings of the third international Symposium on Experimental Robotics*, Kyoto, Japan, 1993

[21] J.P. Nordin, A Compiling Genetic Programming System that Directly Manipulates the Machine Code, in: K. Kinnear (ed.), *Advances in Genetic Programming*, MIT Press, Cambridge, MA, 1994

[22] J.P. Nordin, W. Banzhaf, Complexity Compression and Evolution, in: *Proc. 6th Int. Conference on Genetic Algorithms, ICGA-95*, L. Eshelman (ed.), Morgan Kaufmann, San Mateo, 1995

[23] J.P. Nordin, W. Banzhaf, Evolving Turing Complete Programs for a Register Machine with Self Modifying Code, in: *Proc. 6th Int. Conference on Genetic Algorithms, ICGA-95*, L. Eshelman (ed.), Morgan Kaufmann, San Mateo, 1995

[24] P. Nordin, W. Banzhaf, *A Genetic Programming System Evolving Obstacle Avoidance Behavior and Controlling a Miniature Robot in Real Time*, SysReport 4/95, Department of Computer Science, University of Dortmund

[25] P. Nordin, W. Banzhaf, *Real Time Evolution of Behavior and a World Model for a Miniature Robot using Genetic Programming*, SysReport 5/95, Department of Computer Science, University of Dortmund

[26] P. Nordin, W. Banzhaf, An On-Line Method to Evolve Behavior and to Control a Miniature Robot in Real Time with Genetic Programming, *Adaptive Behavior*, in press

[27] M. Olmer, P. Nordin and W. Banzhaf, Evolving Real-Time Behavioral Modules for a Robot with GP, *Proc. Sixth International Symposium on Robotics And Manufacturing (ISRAM-96)*, Montpellier, France, 1996, in press

[28] I. Rechenberg, *Evolutionsstrategie 94*, Holzmann-Froboog, Stuttgart, 1994 (2nd. ed.)

[29] C. W. Reynolds, Not Bumping into Things, in: Notes for the *SIGGRAPH'88 course Developments in Physically-Based Modeling*, ACM-SIGGRAPH, 1988

[30] C.W. Reynolds, Evolution of Obstacle Avoidance Behavior, in: *Advances in Genetic Programming*, K. Kinnear, (ed.), MIT Press, Cambridge, MA, 1994

[31] H.P. Schwefel, *Evolution and Optimum Seeking*, Wiley, New York, 1995 (2nd. ed.)

[32] R. Zapata, P. Lepinay, C. Novales, P. Deplanques, Reactive Behaviors of Fast Mobile Robots in Unstructured Environments: Sensor-based Control and Neural Networks, in: *From Animals To Animats 2: Proceedings of the Second International Conference on simulation of Adaptive Behavior*, J.Meyer and S. Wilson (eds.), MIT Press, Cambridge, MA, 1993