Evolving real-time behavioral modules for a robot with GP

Markus Olmer Peter Nordin Wolfgang Banzhaf
Fachbereich Informatik
Universität Dortmund
44221 Dortmund, GERMANY
email: olmer,nordin,banzhaf@ls11.informatik.uni-dortmund.de

Abstract

In this paper we demonstrate an efficient method which divides a control task into smaller sub–tasks. We use a Genetic Programming system that 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. The Swiss miniature robot Khepera is employed as the experimental platform. Results are presented which show that the robot is indeed able to evolve both the control algorithms for the different lower–level tasks and the strategy for the selection of tasks. The evolved solutions also show robust performance even if the robot is lifted and placed in a completely different environment or if obstacles are moved around.

INTRODUCTION

Robotics plays an increasingly important role in our society and its potential for the future is tremendous. Especially autonomous robots have a large potential in, for instance, medical applications, hostile environments, exploration tasks, or in the aiding of disabled people. An autonomous robot needs to be flexible enough to cope with an environment that is incompletely known and imperfectly modeled, when using sensors which might be noisy or even defective. Plainly, the number of possible errors in a real–life task is nearly infinite.

A useful strategy to create robust behavior is to divide complex actions into *action primitives* which only perform specialized tasks like avoiding obstacles or moving ahead, and then combine them again for the creation of higher *levels of competence* (Brooks [1]). Our approach uses a learning robot. The goal here is to have a robot that:

- can learn certain action primitives and a control structure and combine these to create complex behavior.
- is able to do this learning in real time, on–line.
- is fault-tolerant recovering after recognizing an error, always being able to act in a robust way in an otherwise unknown or only partially known environment.
- operates with an extendable system which adapts to changes, such as new action primitives.

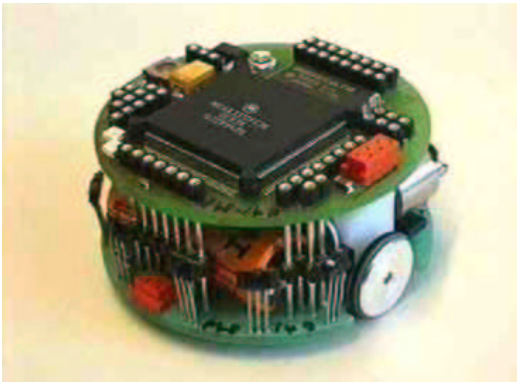

Figure 1: The Swiss robot system KHEPERA. The diameter is 6 cm and its height is 4 cm, depending on how many expansion turrets are installed (none in this Figure).

For this purpose we implemented a variant of a genetic programming system called a compiling genetic programming system (CGPS) which uses different fitness functions for all its tasks. It creates binary machine code programs that fulfill the respective fitness criteria.

COMPILING GENETIC PROGRAMMING

The Compiling Genetic Programming System [5, 7, 8] is based on the Genetic Programming paradigm of John Koza [4]. The system generates machine code for SUN workstations by directly manipulating binary code with genetic operators. It has been shown to have several advantages when compared to another adaptive technique, neural networks. It is frequently faster, requires less memory and has good generalization capabilities [6]. It also produces output in a more symbolic form which is beneficial when trying to analyse why the robot behaves in a certain way. The system already previously proved to work for the complex task of a robot evolving obstacle avoiding behavior [9].

The method uses a stochastic sampling of the environment [9, 10] where randomly selected individuals compete against each other in a tournament selection procedure in *different* situations which could result in an "unfair" competition. However, in the long term the better performing individuals survive.

More precisely, the GP system randomly takes four individuals of the same population [1] and performs the following steps on every individual:

- Feed sensory values to one of the individuals in the tournament.
- Execute the individual and store the resulting value.
- Compute fitness.

After all steps the system computes the winners of the matches of individual 1 against individual 2 and of individual 3 against individual 4. Mutated and recombined versions of the winners then replace the losers as new individuals. This execution cycle is identical for all the action primitives and for the action selection mechanism. However, the action selection mechanism operates on a ten times slower time scale.

[1] We use separate populations of individuals with different fitness criteria for each of the tasks

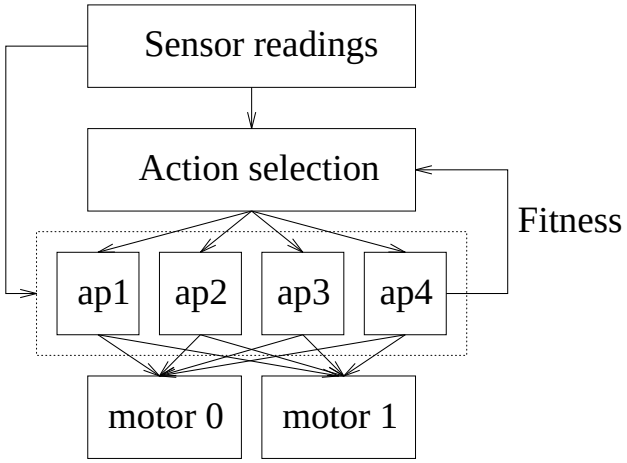

Figure 2: Data flow and architecture of the robot operating system.

THE ROBOT EXPERIMENT

The experimental environment consists of a black floor with moveable white walls so it can be changed easily to test the features and capabilities of the system. Walls can be rearranged and fixed with velcro fly. The walls provide good reflection for the robot's sensors. The GP system runs fully on the Khepera robot (see Figure 1, [3]) which has a simple multitasking operating system onboard.

The goal of the controlling GP system is to evolve several simple behaviors in a sense–think–act context. The controlling algorithm has a small population size, typically less than 40 individuals. The individuals of every species use selected values 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 last computed.

The robot's GP system possesses five populations for the action primitives and the control structure:

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

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

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

FIND DARK : the robot searches for a dark gradient to head towards.

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

The parameters for the GP system are shown in Table 1.

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. Again 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. The genetic operators ensure that the offspring

Parameters	Values
Objective:	Evolve action primitives
Constants (Terminal set):	Integers form 0 to 8191
Instructions (Function set):	ADD, SUB, MUL, SHL, SHR, XOR, OR, AND
Input registers:	3 to 6
Output registers:	1 (sometimes interpreted as a vector)
Maximum population size:	36
Crossover probability:	100 percent
Mutation probability:	5 percent
Selection:	Tournament selection, size 4
Termination criterion:	None
Maximum number of generations:	infinite
Maximum number of instructions:	256

Table 1: Control parameters of the robot experiments with the CGPS system

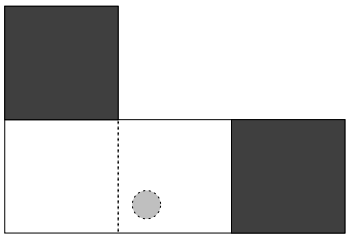

Figure 3: The experimental setup for the robot. The round spot is the starting point for the robot and the marked areas are dark places for the robot to hide in.

is syntactically correct and do not contain any unwanted machine code instructions or constants. Afterwards the GP system updates the selection population. The overall architecture is depicted in Figure 2. This mechanism continues forever to allow the robot to adapt continuously to new situations.

PERFORMANCE OF THE SYSTEM

Performance of the system has been measured in different experiments with the robot. The tasks were: collision avoidance, wall following and hiding in the dark. For every task six experiments were done, the best and the worst have not been counted. The performance of the evolved collision avoidance behavior and the action selection mechanism underlying it is shown in Table 2. The robot starts to show effective collision avoidance behavior after 800 tournament cycles. The second experiment dealt with the robot's capability to recognize dark areas in its environment and to stay there. Here, the performance measurement is the time the robot spends in the dark areas of its environment which is shown in Figure 3. Results are shown in Table 3.

The high values in the beginning of the evolution are caused by random movement of the robot in a dark area. The robot stays in the dark for a long time when it accidently keeps touching the wall while trying to evolve obstacle avoidance. In the end, the time spent in the dark constantly increases due to the success of the GP system. The next task for the robot is to follow the walls of the experimental environment. It comprises a combination of obstacle avoiding and obstacle seeking behavior. This time, we measure the number of times when the robot exits a narrow space (4 cm) between the robot and the wall.

Tournament cycles	Average Collisions	Best rate	Worst rate
0	30.25	22	38
100	17.75	9	20
200	13.25	4	24
300	14.25	6	20
400	12.75	5	12
500	7.50	6	9
600	10.25	7	16
700	8.00	2	11
800	6.00	3	8
900	5.75	3	11
1000	6.75	3	10

Table 2: The number of collisions during 100 tournament cycles, phase 0 only uses random programs. In two out of eight random tests the robot totally locked up in a corner. The learned behavior appears to be stable after 1000 cycles.

Tournament cycles	Average score
0	8
100	26
200	26.5
300	33.25
400	72.25
500	28.75
600	28
700	44.75
800	53.5
900	52.25
1000	68.25

Table 3: Robot scores during the finding–dark–behavior. The maximum score is 120 points where one point is scored for every second spent in the dark. Approximately 120 seconds is the reachable maximum but the time the robot needs to find the dark is between 10 and 15 seconds. Phase 0 only uses random programs.

RESULTS

After a short period of approximately 4-7 minutes, the robot has learned its action primitives and shows robust behavior in its test environment. When the robot is placed into a different environment, it takes about two additional minutes until it has adapted its behavior, e. g., by adjusting to new reflection properties of the walls. With the action primitives performing better the selection mechanism improves also. This requires, however, significantly larger amounts of time, because the fitness values of its actions are to be fed to the selection tournament: the selection mechanism can never get a good score without the actions receiving good fitness values in turn.

Our experiments have shown that GP can be a useful approach to robot control. It is, however, not error–free in every situation, since the robot sometimes touches a wall before converging again after a change in the environment. One of the advantages of this system is its property of easily adapting to new situations. A change in the environment does not cause the robot to need new instructions. Another advantage is the system's capability of generalizing what it has seen – the mechanisms emerging in its behavior would have taken a very long time to be modelled and implemented by a programmer. These results indicate the feasibility of GP in robotics, for reasons of speed, hardware needs, adapability and robustness.

FURTHER WORK

Presently, the robot is trained on tasks with low complexity. Our next steps will be to implement more action primitives in order to test the method more thoroughly. With more action primitives, more complex behavior is expected to emerge.

In the present version, the system does not use a representation or a model of the world, following the notion that "...the world is its own best model. It is always up to date. It always contains every detail there to be known. The trick is to sense it appropriately and often enough" ([2]). Whereas this is true for simple actions, Steels states: "Autonomous agents without internal models will always be severely limited" ([11]). For instance, a world model would be essential where path planning is needed. Also, by adding memory access to our GP system, in addition to the memory provided by the populations of programs, mapping of the world could become possible.

References

[1] Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2:14–23, 1986.

[2] Rodney A. Brooks. Why elephants don't play chess. *Robotics and Autonomous Systems*, 6:3–15, 1990.

[3] K-Team. *Khepera User Manual.* EPFL, Lausanne, 1994.

[4] John R. Koza. *Genetic Programming - On the Programming of Computers by the Means of Natural Selection.* MIT Press, Cambridge Mass., 1992.

[5] P. Nordin. Cgps - a compiling genetic programming system that directly manipulates the machine-code. In *Advances in Genetic Programming*, pages 311–331. MIT Press, Cambridge, Mass., 1994.

[6] P. Nordin. Comparison of a compiling genetic programming system versus a connectionist approach. In *Handbook of Evolutionary Computation.* Oxford University Press, to appear, 1997.

[7] P. Nordin and W. Banzhaf. Complexity compression and evolution. In L. Eshelman, editor, *Proceedings of the Sixth International Conference of Genetic Algorithms*, pages 310—317, San Mateo, CA, 1995. Morgan Kaufmann.

[8] P. Nordin and W. Banzhaf. Evolving turing-complete programs for a register machine with self-modifying code. In L. Eshelman, editor, *Proceedings of the Sixth International Conference of Genetic Algorithms*, pages 318—325, San Mateo, CA, 1995. Morgan Kaufmann.

[9] P. Nordin and W. Banzhaf. Genetic programming controlling a miniature robot in real time. Technical Report 4/95, Department of Computer Science, University of Dortmund, 1995.

[10] P. Nordin and W. Banzhaf. Real time evolution of behavior and a world model for a miniature robot using genetic programming. Technical Report 5/95, Department of Computer Science, University of Dortmund, 1995.

[11] Luc Steels. Exploiting analogical representations. *Robotics and Autonomous Systems*, 6:71–88, 1990.