Genetic Programming Controlling a Miniature Robot

Peter Nordin* Wolfgang Banzhaf†
Fachbereich Informatik
Universität Dortmund
44221 Dortmund, Germany

August 14, 1995

Abstract

We have evaluated the use of Genetic Programming to directly control a miniature robot. The goal of the GP-system was to evolve real-time obstacle avoiding behaviour from sensorial data. The evolved programs are used in a sense-think-act context. We employed a novel technique to enable real time learning with a real robot. The technique uses a probabilistic sampling of the environment where each individual is tested on a new real-time fitness case in a tournament selection procedure. The fitness has a pain and a pleasure part. The negative part of fitness, the pain, is simply the sum of the proximity sensor values. In order to keep the robot from standing still or gyrating, it has a pleasure componentton fitness. It gets pleasure from going straight and fast. The evolved algorithm shows robust performance even if the robot is lifted and placed in a completely different environment or if obstacles are moved around.

*email:nordin@ls11.informatik.uni-dortmund.de
†email: banzhaf@ls11.informatik.uni-dortmund.de

1 Introduction

We have evaluated the use of Genetic Programming to control a miniature robot. To use a genetic process as the architecture for mental activities could, at first, be considered awkward. As far as we know today, genetic information processing is not directly involved in information processing in brains, but the idea of genetics as a model of mental processes is not new. Just 15 years after Darwin published *The origin of Species*, in 1874, the American psychologist William James argued that mental processes operate in a Darwinian manner. He suggested that ideas might somehow "compete" with one another in the brain leaving only the best or fittest. Just as Darwinian evolution shaped a better brain in a couple of million years, a similar Darwinian process operating within the brain might shape intelligent solutions to problems on the time scale of thought and action. This allows "our thoughts to die instead of ourselves". More recently, selectionist approaches to learning have been studied in detail by Gerald Edelman and his collaborators (see [1] and refernces therein). The use of an evolutionary method to evolve controller architectures has been reported pre-

dynamic recurrent neural nets [2], [3]. Several experiments have also been performed were a controller program has been evolved directly through genetic programming [4], [5], [6].
Previous experiments, however, with genetic programming and robotic control have been performed with a simulated robot and a simulated environment. In such a set-up, the environment and the robot can be easily reset to an initial state in order to ensure that each individual in the population is judged starting from the same state. Apart from being practically infeasible for a real robot, this method could result in over-specialization and failure to evolve a behaviour that can generalize to unseen environments and tasks. To overcome this last problem noise is sometimes artificially added to the simulated environment.
In our experiments we use a real robot trained in real time with actual sensors. In such an environment, the system has to evolve robust controllers because noise is present everywhere and the number of real-life training situations is infinite. In addition, it is highly impractical to reset the robot to a predefined state before evaluating a fitness case. Consequently, we had to devise a new method which ensures learning of behaviour while the environment is probabilistically sampled with new real-time fitness cases for each individual evaluation.

2 The Khepera Robot

Our experiments were performed with a standard autonomous miniature robot, the Swiss mobile robot platform Khepera. 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: 2-5 cm. The robot is also equipped with a Motorola 68331 micro-controller which can be connected to a SUN workstation via serial cable.
It is possible to control the robot in two ways. The controlling algorithm could be run on the workstation, with data and commands communicated through the serial line. Alternatively, the controlling algorithm is cross-compiled on the workstation and down-loaded to the robot which then runs the complete system in a stand-alone fashion. At present, the GP-system is run on the workstation, but we plan to port it and down-load it to the robot. The micro controller has 256 KB of RAM and a large ROM containing a small operating system. The operating system has simple multitasking capabilities and manages the communication with the host computer.
The robot has several extension ports where peripherals such as grippers and TV cameras might be attached.

3 Objectives

The goal of the controlling GP system is to evolve obstacle avoiding behaviour in a sense-think-act context. The system operates real-time and aims at obstacle avoiding behaviour from real noisy sensorial data. For a more general description, definition and discussion of the problem domain see, [7], [8], [9], [10].
The controlling algorithm has a small population size, typically less than 50 individuals. The individuals use six values from the sensors as input and produce two output values that are transmitted to the robot as motor speeds. Each individual program does this manipulation independent of the others and thus stands

for an indivudual behaviour of the robot if invoked to control the motors. The resulting variety of behaviours does not have to be generated artificially, e.g. for explorative behaviour, it is always there, since a population of those individuals is processed by the GP system.

3.1 Training Environment

The robot was trained in two different environments. The first environment was a simple rectangular box. The dimensions of the box were 30 cm × 40 cm. The surface transmitted high friction to the wheels of the robot. This caused problems in the initial stages of training. Before the robot learned a good strategy it kept banging into the walls trying to "go through the walls". The high friction with the surface consequently stressed the motors.
We also designed a second, more complex environment. This larger environment is about 70 cm × 90 cm. It has an irregular boarder with different angles and four deceptive dead-ends in each corner. In the larger open area in the middle, loose obstacles can be placed. The friction between wheels and surface is considerably lower, enabling the robot to slip with its wheels during a collision with an obstacle. There is an increase in friction with the walls making it hard for the circular robot to turn while in contact with a wall.

4 The Evolutionary Algorithm

The GP-system is a steady state tournament selection algorithm [5], [11] with the following execution cycle:

1. Select k members for tournament.
2. For all members in tournament do:
   (a) Read out proximity sensors and feed the values to one individual in the tournament.
   (b) Execute the individual and store the resulting robot motor speeds.
   (c) Send motor speeds to the robot.
   (d) Sleep for 400ms to await the results of the action.
   (e) Read the proximity sensors again and compute fitness, see below.
3. Perform tournament selection.
4. Do mutation and crossover.
5. Goto step 1.

4.1 Fitness calculation

The fitness has a pain and a pleasure part. The negative contribution to fitness, called pain, is simply the sum of all proximity sensor values. The closer the robot's sensors are to an object, the more pain. In order to keep the robot from standing still or gyrating, it has a positive contribution to fitness, called pleasure, as well. It receives pleasure from going straight and fast. Both motor speed values minus the absolute value of their difference is thus added to the fitness.
Let p_i be the values of the proximity sensors ranging from $0 - 1023$ where a higher value means closer to an object. Let m_1 and m_2 be the left and right motor speeds resulting from an execution of an individual. The fitness value can then be expressed more formally as:

$$f = \sum p_i + \mid 15 - m_1 \mid + \mid 15 - m_2 \mid + \mid m_1 - m_2 \mid \quad (1)$$

4.2 Implementation

The Evolutionary Algorithm we use in this paper is an advanced version of the CGPS described in [12], composed of variable length strings of 32 bit instructions for a register machine. The system has a linear genome. Each node in the genome is an instruction for a register machine. The register machine performs arithmetic operations on a small set of registers. Each instruction might also include a small integer constant of maximal 13 bits. The 32 bits in the instruction thus represents simple arithmetic operations such as "a=b+c" or "c=b*5". The actual format of the 32 bits corresponds to the machine code format of a SUN-4 [13], which enables the genetic operators to manipulate binary code directly. For a more thorough description of the system and its implementation, see [14].

The set-up is motivated by fast execution, low memory requirement and a linear genome which makes reasoning about information content less complex. This system is also used with the future micro-controller version in mind.

The system is a machine code manipulating GP system that uses two-point string crossover. A node is the atomic crossover unit in the GP structure. Crossover can occur on either or both sides of a node but not within a node. Because of our particular implementation of GP works with 32 bit machine code instructions , a node is a 32 bit instruction.

Mutation flips bits inside the 32-bit node. The mutation operator ensures that only the instructions in the function set and the defined ranges of registers and constants are the result of a mutation.

The function set used in these experiments are all low-level machine code instructions. There are the arithmetic operations ADD, SUB and MUL. The shift operations SLL and SLR and finally the logic operations AND, OR and XOR. All these instructions operate on 32-bit registers.

Table 1 gives a summary of the problem according to the conventions used in [4].

5 Results

Interestingly, the robot shows exploratory behaviour from the first moment. This is a result of the diversity in behaviour residing in the first generation of programs which has been generated randomly. Naturally, the behaviour is erratic at the outset of a run.

During the first minutes, the robot keeps colliding with different objects, but as time goes on the collisions become more and more infrequent. The first intelligent behaviour usually emerging is some kind of backing up after a collision. Then the robot gradually learns to steer away in an increasingly more sophisticated manner.

After about 20 minutes, the robot has learned to avoid obstacles in the rectangular environment almost completely. It has learned to associate the values from the sensors with their respective location on the robot and to send correct motor commands. In this way the robot is able, for instance, to back out of a corner or turn away from an obstacle at its side. Tendencies toward adaption of a special path in order to avoid as many obstacles as possible can also be observed.

The robot's performance is similar in the second, more complex environment. The convergence time is slower. It takes about 40-60 minutes, or 200-300 generation equivalents, to evolve a good obstacle avoiding behaviour. The reason for the slower convergence time could be: less frequent collisions in the larger environment, the slippery surface, the high friction in collisions with walls and/or the less

Table :	
Objective :	Obstacle avoiding behaviour in real-time
Terminal set :	Integers in the range 0-8192
Function set :	ADD, SUB, MUL, SHL, SHR, XOR, OR, AND
Raw and standardized fitness :	Pleasure subtracted from pain value desired value
Wrapper :	None
Parameters :	
Maximum population size :	30
Crossover Prob :	90%
Mutation Prob :	5%
Selection :	Tournament Selection
Termination criteria :	None
Maximum number of generations:	None
Maximum number of nodes:	256 (1024)

Table 1: Summary of parameters used during training.

regular and more complex environment.

6 Future Work and Discussion

We have demonstrated that a GP system can be used to control an existing robot in a real-time environment with noisy input. The evolved algorithm shows robust performance even if the robot is lifted and placed in a completely different environment or if obstacles are moved around. We believe that the robust behaviour of the robot partly could be attributed to the built in generalisation capabilities of a genetic programming system [15].
Our next immediate goal is to cross-compile the GP-system and run it with the same set-up on the micro-controller on-board the robot. This would demonstrate the applicability of Genetic Programming to control tasks on low-end architectures. The technique could then potentially be applied to many one-chip control applications in, for instance, consumer electronics devices etc.
Another further extension to the system would be to eliminate the 400ms delay time of sleeping, during which the system is waiting for the result of its action. This could be achieved by allowing the system to memorize previous stimulus-response pairs and by enabling it to self-inspect memory later on in order to learn directly from past experiences without a need to wait for results of its actions. We anticipate the former to speed up the algorithm by a factor of at least 1000. The latter method would probably speed up the learning of behaviour by a comparably large factor.

Acknowledgement

One of us (P.N.) acknowledges support by a grant from the Ministerium für Wissenschaft und Forschung des Landes Nordrhein-Westfalen under contract I-A-4-6037-I

References

[1] Edelman G. (1987) *Neural Darwinism*, Basic Books, New York

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

[3] Harvey I., Husbands P. and Cliff D.(1993) Issues in evolutionary robotics, in: *From Animals To Animats 2: Proceedings of the Second International Conference on simulation of Adaptive Behaviour*, Meyer and Wilson (eds.), MIT Press, Cambridge, MA

[4] Koza, J. (1992) *Genetic Programming*, MIT Press, Cambridge, MA

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

[6] Handley S. (1994) The automatic generation of Plans for a Mobile Robot via Genetic Programing with Automatically defined Functions, in: *Advances in Genetic Programming*, K. Kinnear, Jr. (ed.), MIT Press, Cambridge, MA

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

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

[9] Zapata R., Lepinay P., Novales C. and Deplanques P. (1993) Reactive Behaviours 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 Behaviour*, Meyer and Wilson (eds.), MIT Press, Cambridge, MA

[10] Braitenberg V. (1984) *Vehicles*, MIT Press, Cambridge, MA.

[11] Syswerda G. (1991) A study of Reproduction in Generational Steady-State Genetic Algortihms, in: *Foundations of Genetic Algorithms*, Rawlings G.J.E. (ed.), Morgan Kaufmann, San Mateo, CA

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

[13] The SPARC Architecture Manual,(1991), SPARC International Inc., Menlo Park, CA

[14] Nordin J.P. and Banzhaf W. (1995) Evolving Turing Complete Programs for a Register Machine with Self-Modifying Code, in: *Proceedings of Sixth International Conference of Genetic Algorithms, Pittsburgh, 1995*, L. Eshelman (ed.), Morgan Kaufmann, San Mateo, CA

[15] Nordin J.P. and Banzhaf W. (1995) Complexity Compression and Evolution, in

Proceedings of Sixth International Conference of Genetic Algorithms, Pittsburgh, 1995, L. Eshelman (ed.), Morgan Kaufmann, San Mateo, CA