Available as: **Technical Report: SysReport 4/95,**
Department of Computer Science,
University of Dortmund
44221 Dortmund, GERMANY

A Genetic Programming System Learning Obstacle Avoiding Behavior and Controlling a Miniature Robot in Real Time

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

Abstract

One of the most general forms of representing and specifying behavior is by using a computer language. We have evaluated the use of the evolutionary technique of Genetic Programming (GP) to directly control a miniature robot. The goal of the GP-system was to evolve real-time obstacle avoiding behavior from sensorial. 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 using genetic programming. To our knowledge, this is the first use of GP with a real robot. The method 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 robots behavior is evolved without any knowledge of the task except for the feed-back from a fitness function. 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 component to its 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.

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, though the idea of genetics as a model of mental processes is not new. William James, the father of American psychology, argued just 15 years after Darwin published *The origin of Species*, in 1874, that mental processes could operate in a Darwinian manner (William James 1890). He suggested that ideas "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 Edelman 1987) and references therein).
The use of an evolutionary method to evolve controller architectures has been reported previously in a number of variants. Robotic controllers have, for instance, been evolved using dynamic recurrent neural nets (Cliff 1991), (Harvey, Husbands &Cliff (1993).

Genetic programming uses an evolutionary algorithm to evolve computer programs. Several experiments have also been performed were a controller program has been evolved directly through genetic programming (Koza 1992), (Reynolds 1994), (Handley 1994).
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 behavior 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 behavior while the environment is probabilistically sampled with new real-time fitness cases for each individual evaluation.

2 Genetic Programming

Evolutionary Algorithms mimic aspects of natural evolution, to optimize a solution towards a defined goal. Darwin's principle of natural selection and survival of the fittest is thought to be responsible for the evolution of all life forms on earth. This principle has been employed successfully on computers over the past 30 years. Different research subfields have emerged such as , Evolution Strategies (Schwefel 1995), Genetic Algorithms (Holland 1975) and Evolutionary Programming (Fogel, Owens & Walsh 1966) indicating that they all mimic various aspects of natural evolution. In recent years, these methods have been applied successfully to a spectrum of real-world and academic problem domains, including robot control as mentioned above.
A comparatively young and growing research topic in this field is Genetic Programming (GP). Genetic Programming uses the mechanisms behind natural selection for *evolution of computer programs*. Instead of a human programmer programming the computer, the computer can self-modify, through genetic operators, a population of programs in order to finally generate a program that solves the defined problem. This technique, like other adaptive techniques, has applications in problem domains where theories are incomplete and insufficient for the human programmer or when there isn't enough time or resources available to allow for human programming.
Methods for GP were suggested as far back as in the 1950s (Friedberg 1958). For various reasons these experiments never were a complete success even if partial results were achieved (Cramer 1985). A breakthrough came when J. Koza formulated his approach based on program individuals as tree structures represented by LISP S-expressions (Koza 1992). His hierarchical subtree exchanging genetic crossover operator guaranteed syntactic closure during evolution. Koza has been instrumental in developing GP and was the first to demonstrate the feasibility of the technique with dozens of applications.

In this motion, genetic programming evolves programs stored as trees in a population by exchanging subtrees during crossover. Selection is performed according to a fitness function. Figure 1 illustrates the crossover method used with genetic programming.

In this paper, however, we use a variant of GP which stores the individuals of the population as binary machine code in memory. The gives a speed-up of several orders of magnitudes. The method is also memory efficient using only 32KB for the GP kernel. The individuals can be stored in an economic way and memory consumption is stable during evolution with no need for garbage collection etc. All these properties make the system ideally suited for real-time control in low-end processor architectures such as one-chip embedded control.

The system uses a linear representation of the program. The crossover operators exchanges blocks of code that are cut out at instruction boundaries. A mutation operator changes the inside of an instruction by mutating constants and register references. Both operators ensure syntactic closure during evolu-

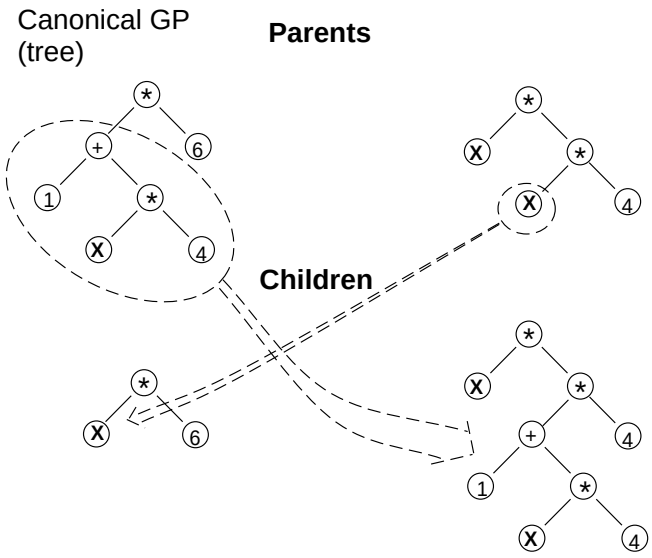

Figure 1: hierarchical crossover in Genetic programming

tion.Figure 2 illustrates the crossover method used in our experiments.

Each node represents an operation between a small set of registers. The structure of the node corresponds to a machine code instruction in the processor and is stored in such a way that they can be executed directly. Below we see how such a program individual may look if printed as a 'C' program.

```
a=s3 + 4;
d=s2 >> s1;
b=s1 - d;
d=s2 + 2;
c=d >> 1;
b=d - d;
a=c - 3;
c=s4 << a;
d=a * 4;
a=a ^ 5;
e=d + s4;
c=b & c;
d=d + d;
c=d | 8;
d=d * 10;
b=e >> 6;
d=b & 0;
```

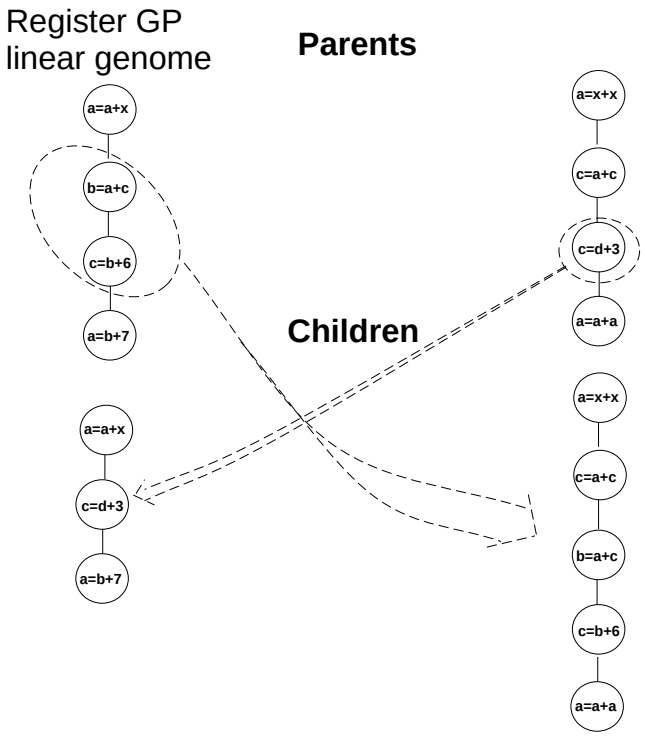

Figure 2: Linear genome and crossover used in this paper.

```
e=c >> b;
motor2=a | e;
c=d | 7;
motor1=c * 9;
c=e & e;
```

Each individual is composed of simple instructions (program lines) between variables and input and output parameters. See section 5 for further details of the evolutionary algorithm, its execution cycle and the implementation.

3 The Khepera Robot

Our experiments were performed with a standard autonomous miniature robot, the Swiss mobile robot platform Khepera (Mondada, Franzi & Ienne 1993). 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, we have two version of the system. In one the controlling GP-system is run on the workstation, and in the other the system is downloaded and run autonomously on the micro-controller of the robot.
The micro-controller has 256 KB of RAM and a large ROM containing a small operating system. The operating system has simple multi-tasking capabilities and manages the communication with the host computer.
The robot has several extension ports where peripherals such as grippers and

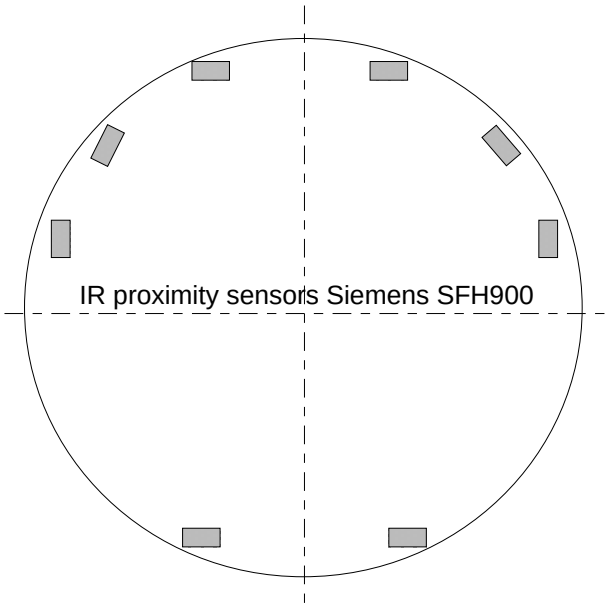

Figure 3: Position of the IR proximity sensors.

TV cameras might be attached.

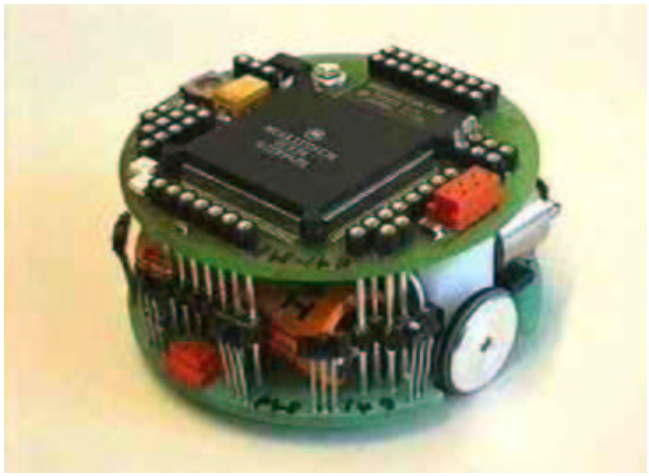

Figure 4: The Khepera Robot

4 Objectives

The goal of the controlling GP system is to evolve obstacle avoiding behavior in a sense-think-act context. The system operates real-time and aims at obstacle avoiding behavior from real noisy sensorial data. The sensorial data come from eight proximity sensors. See (Reynolds 1988), (Mataric 1993), (Zapata et. al. 1993), (Braitenburg 1984) for a discussion of this problem domain.

Symbolic regression is the procedure of inducing a symbolic equation fitting numerical data. Genetic programming is ideal for symbolic regression and most GP applications could be reformulated as a variant of symbolic regression. In this obstacle avoiding application we would like to approximate a function that takes the sensor values as input vector and returns a vector of two motor speeds:

$$f(s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8) = \{m_1, m_2\} \tag{1}$$

The evolved programs are true functions in our experiments because no side-effects are allowed. It would be possible to evolve programs that can store information in memory for later use, but we have chosen to limit the experiment to side effect free programs.

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 which are transmitted to the robot as motor speeds. Each

individual program does this manipulation independent of the others and thus stands for an individual behavior of the robot if invoked to control the motors. The resulting variety of behaviors does not have to be generated artificially, e.g. for explorative behavior, it is always there, since a population of those individuals is processed by the GP system.

4.1 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 $\times$ 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 $\times$ 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.

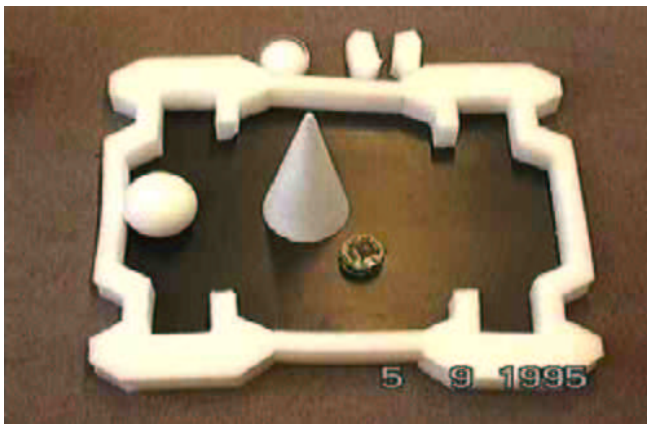

Figure 5: The Training Environment

5 The Evolutionary Algorithm

The population of programs are initialized with random content at the beginning of training. A simple tournament is used for the competitive selection of individuals allowed to have offspring. Figure 6 gives a schematic view of the control architecture and the GP-system.

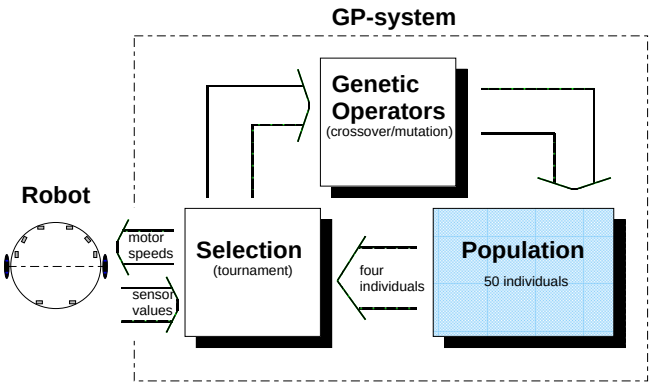

Figure 6: Schematic View of the Control System.

The GP-system with its steady state tournament selection algorithm (Reynolds 1994), (Syswerda 1991) has 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.

Each individual is thus tried against a different real-time fitness case. This could result in "unfair" comparison where individuals have to maneuver in situation with very different possible outcomes. However, our experiments show that over time this probabilistic sampling will even out the random effects in learning and a set of good solutions survive. The convergence time is the same range or lowers as with other GP systems working with simulated environments.

5.1 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 s_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 values of m_1 and m_2 are in the range of zero to 15. The fitness value can then be expressed more formally as:

$$f = \sum s_i + \mid 8 - m_1 \mid + \mid 8 - m_2 \mid + \mid m_1 - m_2 \mid \qquad (2)$$

5.2 Implementation

The Evolutionary Algorithm we use in this paper is an advanced version of the CGPS described in (Nordin 1994), 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 (Sparc 1991), which enables the genetic operators to manipulate binary code directly. For a more thorough description of the system and its implementation, see (Nordin and Banzhaf 1995a).

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 thus a presumption for micro-controller version.

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

Table :	
Objective :	Obstacle avoiding behavior 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.

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 (Koza 1992).

6 Results

Interestingly, the robot shows exploratory behavior from the first moment. This is a result of the diversity in behavior residing in the first generation of programs which has been generated randomly. Naturally, the behavior 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 behavior usually emerging is some kind of backing up after a collision. Then the robot gradually learns to steer away in an increasingly more sophist-

icated 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 adoption 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 behavior. 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 regular and more complex environment.
Figure 7 show how the number of collisions per minute diminishes as the robot learns and the population becomes dominated by good control strategies.

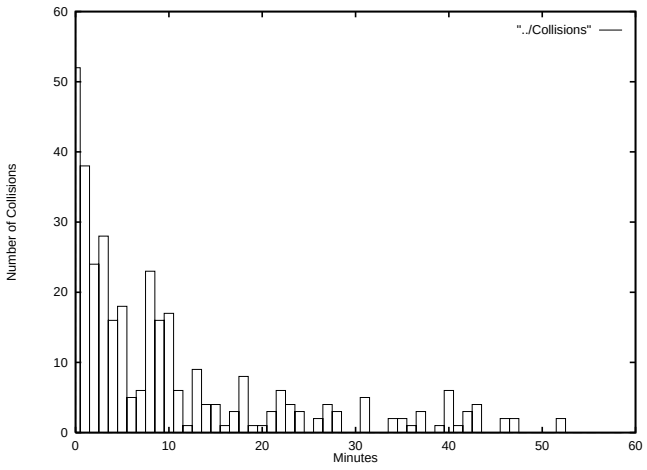

Figure 7: The number of collisions per minute in a typical training run with the environment in Figure 5

The moving robot gives the impression of performing a very complex behavior. Its behavior gives association to a bug or an ant exploring an environment with irregular small moves around the objects. Data such as Figure 8 also suggest that the evolved behavior is complex. It is therefore hard to classify the different kinds of behavior the robot demonstrates during evolution but we found a few distinguishable classes:

1. The initial wall-bumping behavior. This is the childhood of the robot

were it must make mistakes in order to learn for the rest of its life. It tries several impossible strategies such as trying to force it self through the walls

2. The backing–up behavior. This is the first effective technique that allows the robot to avoid obstacles. However, this technique has several drawbacks. It will risk to place the robot in the same situation if it moves forward again.

3. The reflection behavior. Here the robot gradually turns away from an obstacle as it approaches it. It looks as if the robot bounces like a ball at something invisible close to the obstacle. This behavior gives a minimum speed change in the robot's path.

4. The sniffing behavior. The robot stops immediately as it just senses an object. It then starts to sniff around by very small movements to determine which way will be the most favorable to go.

All these behaviors can be distinguished but the also appear mixed and as part of very complex movements.

6.1 Visualizations of results

We have used different visualization techniques to plot the relationship between sensor input and motor output, in order to illustrate the control strategies of the different evolved programs. The actual control function takes eight values as input (the sensors) and produces two values as output (the motor speeds) It thus has a too high dimension for directly visualizing it in a diagram. Instead we simulate a uniform stimulus on the left hand side and the right hand side of the robot. The output of an individual program in the population is examined and we plot the difference between left motor speed and right motor speed. This difference could be interpreted has the direction to which the robot steers. Figure 8 shows an actual example of a plot illustrating the difference in motor speeds depending on stimulus of the sensors.
A simple control strategy that would avoid obstacles to some extent would be a plane sloping from the upper left corner of the diagram down to the lower right corner of the diagram, as in Figure 9. If the robot is controlled by this algorithm it would steer more to the right the more the sensors on the left side are stimulated. Many plots from actual evolved controlling programs show similarities to this plane. For instance, the program behind Figure 10 and Figure 11. Here the diagram is more complex but the slope from upper left to lower right is evident. A good control strategy should, however, accomplish more than simply steering away. It should for instance have the ability to back away from obstacles and perform more complex maneuvers. In many cases we do not have an explanation for the complex look of these diagrams. We do not know whether they are the product of mere chance or whether the complex features help the individual in the competition to control the robot.

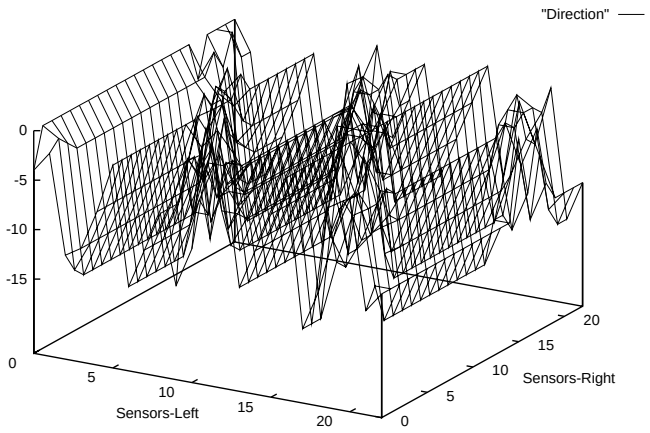

Figure 8: Example the behavior of an evolved control program

Below we see the corresponding individual control program that generates Figure 8 when examined systematically.

```
a=s3 + 4;
d=s2 >> s1;
b=s1 - d;
d=s2 + 2;
c=d >> 1;
b=d - d;
a=c - 3;
c=s4 << a;
d=a * 4;
a=a ^ 5;
e=d + s4;
c=b & c;
d=d + d;
c=d | 8;
d=d * 10;
b=e >> 6;
d=b & 0;
e=c >> b;
motor2=a | e;
c=d | 7;
motor1=c * 9;
c=e & e;
```

This control strategy does not use all of the sensor values, for instance s_0

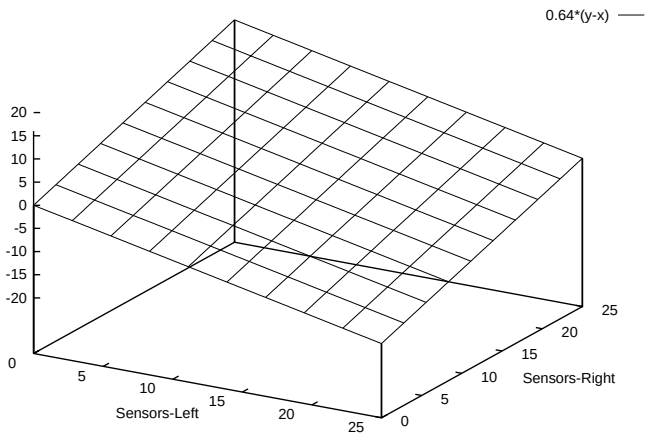

Figure 9: An exmaple of a simple controlling function that would display a simple obstacle avoiding behavior

is missing. The sensors from the back of the robot are not used either in this example. It is typical for solutions of genetic programming to be parsimonious in its output and only use the concepts which really are needed to achieve the defined goal.

The code above has been edited for clarity, sensor and motor speed values are in reality stored in the same register used during calculation. Register "a" contains the value of s_0 when evaluation starts and registers "b" contains the value of s_1 . Correspondingly the motor speed values sent to the motors are what is left in register "a" and "b" at the end of evaluation.

6.2 The autonomous system

It is not completely evident what an autonomous robot is. Some would argue that the autonomy is a property of the controlling algorithm while others would argue that physical autonomy is needed.

In order to try the fully autonomous paradigm, we have ported a special version of the system to the micro-controller. It is possible to download this system via the serial cable to the robot. With the accumulators switched on, the robot can then be disconnected from the workstation and can run completely autonomously. The Motorola 68331 micro-controller then runs the full GP system.

As mentioned earlier, this micro controller has 256 KB of RAM memory. The kernel of the GP system occupies 32 KB and each individual 1KB, in this setup. The complete system with 50 individual then occupies 82 KB which is

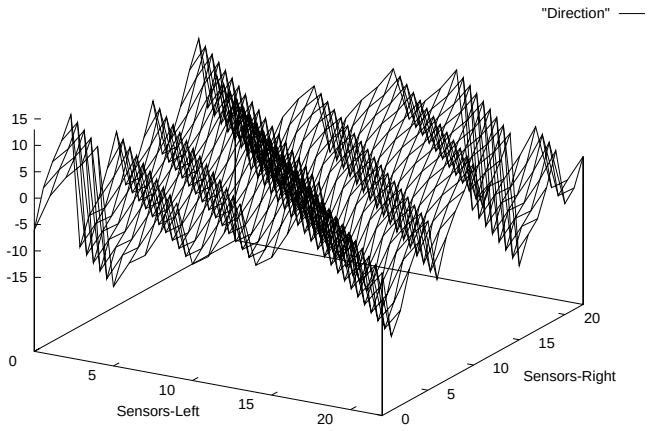

Figure 10: Example of controlling function with similarities with the plane in Figure 9.

well within the limits of the on-board system.

This demonstrates that the compactness of the compiling GP system enables relatively powerful solutions in weak architectures such as those used in embedded control.

7 Conclusions and Future Work

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 behavior of the robot partly could be attributed to the built in generalization capabilities of a genetic programming system (Nordin and Banzhaf 1995b).
We have also cross-compiled the GP-system and run it in the same set-up on the micro-controller on-board the robot. This demonstrates the applicability of Genetic Programming to control tasks on low-end architectures. The technique could potentially be applied to many one-chip control applications in, for instance, consumer electronics devices etc.
A 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

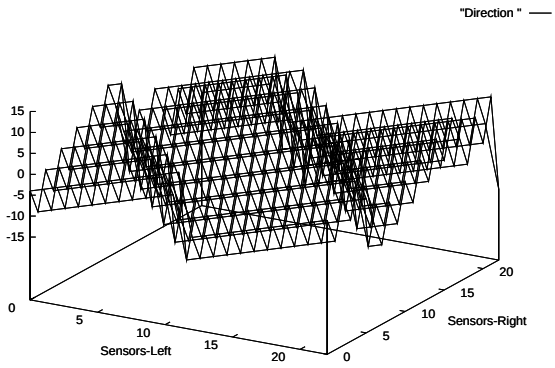

Figure 11: Example of controlling function.

least 1000. The latter method would probably speed up the learning of behavior by a comparably large factor.

Acknowledgment

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] James W. (1890) The principles of psychology Vol.1. Originally published: Henry Holt, 1890.

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

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

[4] 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 Behavior*, Meyer and Wilson (eds.), MIT Press, Cambridge, MA

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

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

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

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

[9] Schwefel, H.-P. (1995) *Evolution and Optimum Seeking*, Wiley,New York

[10] Holland, J. (1975) *Adaption in Natural and Artificial Systems*, Ann Arbor, MI: The University of Michigan Press.

[11] Fogel, L.J., Owens, A.J., Walsh, M.J.,(1966) Artificial Intelligence through Simulated Evolution. Wiley, New York

[12] Banzhaf, W. (1994) Genotype - Phenotype Mapping and Neutral Variation, A case study in Genetic Programming.In *Proc. 3rd. Int. Conf. on Parallel Problem Solving from Nature*, Jerusalem, Y.Davidor, H.P. Schwefel, R. Maenner (Eds.), Springer, Berlin.

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

[14] Cramer, N.L. (1985). A representation for adaptive generation of simple sequential programs. In *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, pp183-187

[15] Mataric M.J.(1993) 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*, Meyer and Wilson (eds.), MIT Press, Cambridge, MA

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

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

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

[19] Mondada, F., Franzi,E., and Ienne, P. (1993) Mobile robot miniaturization. In *Proceedings of the third international Symposium on Experimental Robotics*, Kyoto, Japan.

[20] 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

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

[22] Nordin J.P. and Banzhaf W. (1995a) 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

[23] Nordin J.P. and Banzhaf W. (1995b) Complexity Compression and Evolution, in *Proceedings of Sixth International Conference of Genetic Algorithms, Pittsburgh, 1995*, L. Eshelman (ed.), Morgan Kaufmann, San Mateo, CA