Genetic Evolution of a Neural Network for the Autonomous Control of a Four-Wheeled Robot

Wilfried Elmenreich and Gernot Klingler
Vienna University of Technology
Institute of Computer Engineering
Treitlstrasse 3, 1040 Vienna, Austria
{*wil,gernot.klingler*}*@vmars.tuwien.ac.at*

Research Report 94/2007

Abstract

In this paper we exercise the genetic programming of a Artificial Neural Network (ANN) that integrates sensor vision, path planning and steering control of a mobile robot. The training of the ANN is done by a simulation of the robot, its sensors, and environment. The results of each simulation run are then used to denote the ability for the tested network to operate the robot. After less than hundred evaluations we receive an ANN that is able to navigate the robot around obstacles better than a traditional implementation of sensor-based vision and navigation for the same robot.

1 Introduction

Designing a vision and control software for an autonomous robot is a complex and cumbersome task. The control software has to cope with sensor inaccuracies, actuator modeling, software complexity, and resource constraints of the embedded system. Moreover, the results are typically brittle systems that need to be fine-tuned for a given robot configuration and are often difficult to reuse on different systems or when part of the system changes (e. g., when employing sensors with different behavior).

In order to overcome these problems and to create also innovative solutions to abstract control problems (e. g., "explore the room"), the idea using evolutionary programming by automatically evolving control systems by a genetic algorithm has been proposed by several researchers [18].

A genetic algorithm [19] evaluates a pool of candidates by a given fitness function that estimates the candidates' performance when applied to the intended problem. Then, the best candidates are kept, while the candidates with bad performance are replaced by offsprings or mutations of the pool. Thus, candidates with a high fitness function are evolved over several generations.

A genetic algorithm only works if the candidates are represented in a way that they can easily be mutated and recombined while still retaining a syntactically correct and possible useful program. Therefore, standard programming languages like C or Java do not qualify for this kind of genetic programming. Instead languages like LISP or ANNs are used for program representation.

In this paper we show how to evolve an ANN that instruments the sensors and movement control of an autonomous mobile robot and compare the results to a traditionally implemented control system for the same robot. The intention is to provide a reference implementation for this kind of problem that explains the efficient usage of mutation, crossover, and selection on populations of solutions represented as ANNs. Our results show that the evolutionary approach can be an interesting alternative to a solution from a human designer in terms of performance, design effort, memory footprint, and execution time.

The remaining parts of the paper are organized as follows: Section 2 describes the robot hardware

for which the control system has been designed, explains the specific difficulties for the vision and control system and describes the setup of a simulation environment for this robot. Section 4 describes the ANN and the interfacing between robot hardware and neurons. Section 5 explains the genetic algorithm and discusses the method for the given problem type. Section 7 depicts the results of several experiments carried out in the simulator using the genetic algorithm to evolve an ANN that performs the intended task. Section 8 relates the contributions of this paper to existing approaches in this area. The paper is concluded in Section 9.

2 The robot

The robot, called *Smart Car* consists mechanically of an off-the-shelf four wheeled model car fitted with a wooden mounting board (depicted in Figure 1) of size 0.4 m times 0.3 m. The mounting board hosts several sensors and actuators (see [9] for details) whereof we use only the three Sharp GP2Y0A02 infrared sensors and the actuators for speed and steering control for our experiments. The infrared sensors are mounted on servos, so that the sensor's viewing angle can be adjusted dynamically. The robot features an Ackerman steering at the front axis, basically this means, that the inner wheels are turned at a greater angle when driving into a curve. However, due to this kind of steering, navigation to a specific location is not trivial, since the robot cannot turn on place, but has a turn radius of 0.82 m.

The transducers, i. e., the robot's sensors and actuators, are instrumented by separate microcontrollers performing local transducer-specific instrumentation and signal conditioning while communicating with the main controller via a time-triggered sensor bus. The main controller of the smart car is based on an 8-bit ATmega128 microcontroller running at a clock speed of 14.7456 MHz and featuring 128 KB of Flash ROM and 4KB of RAM memory. Thus, the implementation of the vision and control system is required to have a low resource footprint in terms of memory and computation requirements.

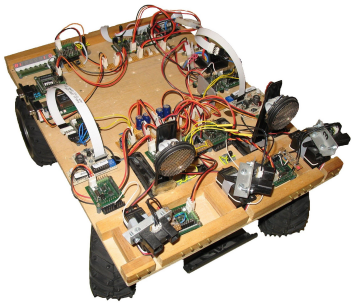

Figure 1: The Smart Car robot

3 Simulation environment

There has been several arguments on simulation vs. real hardware in the literature [13], however we decided to employ simulation for the following reasons:

- When exploring different configurations for the system parameters, there will in overall several hundred thousand evaluation runs taking place, each consisting of the robot being controlled by a different version of the program. This will take considerable time – in our case one evaluation requires the robot moving around for about half a minute. Using simulation, the set-up can be improved and simulations can be sped up and parallelized using standard PCs.
- Most available robots will tend to break down [5] or at least show wear-out behavior on the mechanical components when used for an extended amount of time.
- Maintenance and initialization, e. g., reloading batteries or placing the robot on a defined start position are, unlike in simulations, very difficult to automatize for real-world robots and therefore would require constant support and supervision of a human operator.
- The simulation environment gives easy access to normally unknown parameters like current position or traveling path that could only acquired indirectly by sensor measurements in the real world. Consequently these values support the monitoring and assessment of the evolution process.

The main arguments against simulation lie in the differences between simulation model and reality. However, this problem can be diminished by

designing the simulation model carefully. Given the arguments above, we suggest that simulation should be the first step to involve intelligent behavior in a safe simulation environment that allows to make mistakes (like crashing into walls) and support a fast evaluation of several 100,000 generations using parallelization and a faster-than-real-time simulation mode. After this stage it is still possible to further evaluate the system using its actual hardware. Such a two-stage hybrid approach has been exercised by Nolfi, Miglino and Parisi [14] and the results show that the adaption process of the evolved system to the real world is typically very fast.

The Smart Car has been modeled carefully for the Rossum's Playhouse (RP1) simulator [11]. The model accurately includes the Ackerman steering behavior and a realistic model of the infrared sensors based on measurements published in [16, Chapter 5.1]. Figure 2 depicts the measured behavior of a real Sharp GP2Y0A02 infrared sensor as they are used on the Smart Car. For distances below 20 cm, the sensor does not deliver measurements that can be reliably interpreted.

Distances between 20 and 150 cm can be converted to a distance measurement using the following equation:

$$dist = \frac{a}{x - b} + c$$

where x is the sensor's response and a,b, and c are calibration constants. This conversion is automatically done by the smart sensor. It would have also been possible to feed the ANN with the raw sensor signal, however, a previous evaluation has shown that the arithmetic approach shows better

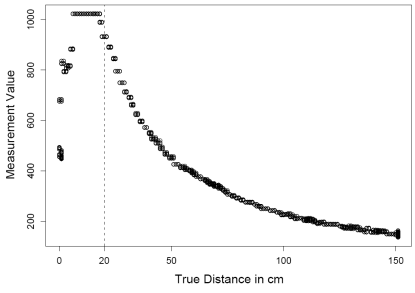

Figure 2: Measured behavior of a GP2Y0A02 distance sensor (from [17])

performance in terms of execution speed and error [10].

The calibrated distance value tends to have a stochastic error of a variance of 2 cm^2 [16, Chapter 5]. Using these data, the behavior of the Sharp GP2Y0A02 infrared sensors has been modeled in RP1.

4 Neural network model

The chosen ANN is a fully connected recurrent time-discrete model similar to that used by Husbands et al. for a similar problem [8]. The neurons are designed for their usefulness in control applications rather than being biologically plausible or easy to analyze. The data types have been optimized for embedded microcontrollers without a floating point unit.

Each neuron is connected to the other neurons via several input connectors. Based on the neurons activation value, the neurons output is forwarded via different connections to all other neurons. The network is fully connected, thus each neuron has also a connection to itself. Each connection is assigned a weight that can be a value between -256.0 and +255.0, represented as fixed point value with a 15 bit mantissa. The output of a neuron is a value between -1.0 and $\frac{127}{128} = 0.992$, represented as fixed point value with a 7 bit mantissa. Additionally, each neuron is assigned a bias value with the same data format as the incoming weights.

The controlling interface to the robot is done by special input and output neurons. The input neurons produce the sensor data on its output independently of the values on their input. The output neurons behave like normal neurons in the network, but their output is also forwarded to the actuators. Therefore, the actuator interface has been linearly scaled to operate within the limits of a neuron's output. For example, the steering actuator has been scaled in order that a value of -1.0 represents the maximum turning angle to the right. The extra nodes which are not characterized as input or output nodes are the so-called hidden neurons.

The network is implemented in software and executed in discrete steps. At each step, each neuron i builds the sum over its bias b_i and its incoming weights $w_j i$ multiplied by the current outputs of the neurons $j = 1, 2, ..., n$ feeding the connections.

Then, an activation function F is applied to value in order to calculate the output of the neuron for step $k+1$:

$$o_i(k+1) = F(\sum_{j=0}^{n} w_{ji} o_j(k) + b_i)$$

where F is a simple linear threshold function

$$F(x) = \begin{cases} -1.0 & \text{if } x \leq -1.0 \\ x & \text{if } -1.0 < x < 0.992 \\ 0.992 & \text{if } x \geq 0.992 \end{cases}$$

The employed network has three input neurons producing the distance value of each sensor (scaled by a factor of $\frac{1}{128}$ in order to fit the data type). Two output nodes are mapped to the steering and motor actuator.

The input neurons are updated periodically in intervals of 100 ms. After each update, two steps of the network are executed. The second step is necessary to enable the hidden neurons to take the current input into account (first step) and contribute to the output neurons (second step).

5 Evolution method

The evolution method involves a genetic algorithm that searches for solutions with a high fitness regarding the intended purpose. Basically, multiple solutions are created using stochastic methods and evaluated in parallel while the best ones are selected for the next generation.

We used a versatile framework [15] for genetic evolution of ANNs that supports mutation, crossover, elite selection, random selection, and co-evolution of multiple populations. Algorithm 1 depicts the basic elements of the genetic algorithm. Each version of an ANN is represented by the weight matrix and the biases of each neuron, which we also call the *genome* of the network.

The selection applies elite selection, that is keeping k_{elite} networks with best scores, and a random selection, where networks with higher scores and greater diversity to the already selected networks have a higher chance of being selected.

The mutation feature applies random variations to the genome, whereas the maximum change rate is proportional to the previous value. Thus, small values are changed by a small random range, while large values may receive larger variations. Thus, the mutation introduces sufficient change on the one hand and is able to make fine variations to small values one the other hand.

Algorithm 1 Genetic algorithm with multiple populations

```
create n networks in m populations and initialize them with random values

for generations
  for p=0 to m
    for i=0 to n
      evaluate network_p, i and store score
    rank networks according to their score (best first)

  for p=0 to m
    for i=0 to n
      keep some networks
      create mutations of kept networks
      create offsprings of kept networks
       create some networks anew and initialize them with random values
```

The design of the crossover operator for ANNs is non-trivial. If not done carefully, offsprings do not inherit the capabilities of their parents but show a new unintended behavior that does not fit the purpose of the task. Since existing crossover operators [4, 7] do not apply to our type of network we designed a new crossover algorithm by taking into account the ideas of the related, not directly applicable, approaches.

First we avoid splitting the genome between the input weights and biases of a neuron, thus a neuron is treated in an atomic way. Our experiments have shown that this approach is advantageous over splitting up the components of a neuron. Second, we try to identifying parts of the network with high connectivity which may act as so-called organs with a certain independence. Then we assign a lower probability of splitting inside organs than splitting between organs.

Random selection is expected to avoid stagnation of the algorithm for local maxima of the fitness value. Another means to overcome this problem is co-evolution, where multiple populations are evolved separately so that the genetic diversity is increased. After a number of generations the

crossover method is allowed to create offsprings from genomes from different populations so that the separately evolved features can be combined in order to find a solution with better performance. In our simulation runs, we selected an interval of 10 generations between inter-population breeding.

6 Experiment setup

Figure 3 depicts the set up of the artificial test environment that was used in the simulations. There are several starting points with different coordinates and headings in order to avoid the situation that a robot becomes trained for only one particular trail. The rectangle in the bottom left in the figure depicts the simulacrum of the robot. The test field covers an area of 6m x 6m.

The general goal of the control application was to make the robot exploring its environment without colliding with the walls. In order to quantify that goal we measure periodically the euclidean distance between the robots starting position and its current position. The maximum distance represents the score of a simulation run. When the robot hits an obstacle or is not able to increase its score during a given time, the evaluation is ended and the current score is returned as the fitness value.

We conducted several experiments for the purpose of evaluating the best parameters and setups for the evaluation regarding appropriate population sizes and number off hidden nodes, evaluation of the effectiveness of mutation and crossover methods, and a performance comparison to the engineered approach based on certainty grid [3] and vector histogram navigation [1].

In order to provide a measure for the quality of the results we have also evaluated the hand-written

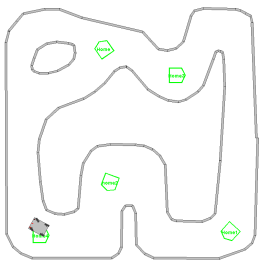

Figure 3: Experimental test environment

vision and navigation system based on the certainty grid/vector field histogram approach using the same framework.

7 Results and evaluation

Table 1 depicts the value of the fitness function after 10, 50, 100 and 200 generations. The best performing configuration was the one with 12 hidden nodes evolved by 2 independent populations. Note that the evolution of this network type, and in general of larger networks is slower than for networks with fewer number of nodes. The genetic algorithm tends also to stagnate at local maxima if the number of parameters to optimize is large, therefore we have employed the parallel evolution scheme for the setups with more than 10 neurons, also it slowed down the evolution process.

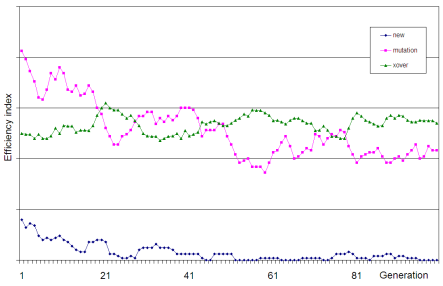

Figure 4: Efficiency of New, Mutation and Crossover operators

Figure 4 depicts the efficiency of the modify operations in our genetic algorithm. The new modifier has its main benefit in the start phase, while the mutation modifier supports the evolution of the population over the whole process. However, after 50 generations, the crossover operator outperforms the other two. Note that this does not mean one can replace the three operators just by the best performing one – the new operator is important since it introduces the biggest diversity, and the mutation and crossover benefit from each other in order to achieve individuals with better fitness.

The most interesting result of this experiment is the performance comparison to the traditionally implemented navigation system. For the given purpose, the ANN approach outperforms the engineered solution after 50 generations.

Table 1: Simulation results of different configurations

	fitness after generation			
network configuration[a]	10	50	100	200
3 input nodes, 4 hidden nodes, 2 output nodes, 15% elite selection, 12% random selection, 5% renewed, 20% mutants, 48% offsprings, 1 populations of 100 nets	148	224	245	312
2 population of 50 nets, 8 hidden nodes	162	175	190	200
2 populations of 50 nets, 12 hidden nodes	134	259	296	356
68% mutants, 0% offsprings	150	184	207	211
Certainty grid/vector histogram approach[b]	160			

[a]The first configuration is the standard setting for the experiments, the lines below only denote the changes with respect to the standard configuration.

[b]This approach does not evolve over generations.

Table 2: Memory requirements for ANN controllers (3input, 12hidden, 2 output nodes) and certainty grid/vector histogram approach

	Program Code (Flash)	Data tables (Flash)	Variables (RAM)
Neural network with fixed point arithmetics	502 Bytes	640 Bytes	36 Bytes
Neural network with floating point arithmetics	3982 Bytes	1280 Bytes	140 Bytes
Certainty grid/vector histogram navigation	23973 Bytes	–	3168 Bytes

We evaluated the fitness of the ANN approach for embedded devices by the example of an implementation in C for the Atmel AVR architecture[1]. Table 2 depicts the resulting memory statistics. As a benchmark for our fixed point integer arithmetics approach, we depict also the values for a floating point version of the ANN. The ANN controller using fixed point integer arithmetic shows a very frugal memory consumption. Note that some 8-bit microcontrollers provide only a few kilobytes of Flash memory and even less SRAM memory, so that a small memory footprint is of great relevance for compact designs without external memory circuitry. The execution time for feeding the measurements into the ANN, performing two steps (see Section 4 for an explanation of the two steps) and forwarding the output to steering and motor control required 1.61 ms in fixed point arithmetic and 109.6 ms in floating point arithmetics on an Atmel ATmega128 running at 14.7456 MHz. The computation time has no measurable jitter since the operations are executed in several loops with a constant number of iterations. The execution time for the fixed point version is considerable fast, since the update time of the sensors of around 70 ms is the limiting factor of the processing speed.

Note, however, that the execution time and Flash memory requirements of the ANN will scale with the square value of the number of neurons. The RAM requirements are direct proportional to the number of neurons (excluding input nodes). For example, a network with 200 neurons will have 200 ms computation time and require 80 kB of Flash memory and 400 Bytes of RAM memory, thus will still fit into one of the larger 8-bit microcontrollers. Networks significantly above that size cannot be reasonably used on 8-bit microcontrollers[2]. However, the limiting factor on the network size is rather on

[1]`http://www.atmel.com/products/avr/`

[2]This applies only to fully connected networks.

the training method than on the time complexity of its implementation, since a genetic algorithm might not be able to cope with a network of that size.

8 Related work

An early inspiration for our work has been given by Braitenberg's vehicles [2]. In these thought experiment Braitenberg proposed simple control designs with very few circuitry (or equivalent program code, but Braitenberg's vehicles typically do not have a central processing unit) that behave in an "intelligent" way.

Genetic algorithms have been applied in several ways to evolve ANNs. Meeden [12] proposes a solution solely based on mutation and selection without crossovers. Also simpler, the learning process is much slower as a consequence and requires several thousand generations in Meedens example. Using our framework we can confirm that there is a significant difference in learning speed between mutation/selection and crossover/mutation/selection approaches.

Floreano and Mondada [6] describe the evolving of a navigation system based on discrete-time recurrent networks, as it is the case in our paper. They successfully evolve a network for controlling a Khepera robot with 12 input nodes, 5 hidden nodes and 2 output nodes over 100 generations. Their approach differs from our task mainly in the mobility of the employed robot, since the Khepera is a two wheeled system that can turn on its place. As a consequence, the control system for our robot requires more hidden nodes in order to perform well.

9 Conclusion

The performance results of the evolved network are significant better than the engineered approach. While saving time on implementation of control strategies on the one hand, considerable effort has to be put into the modeling of the simulation environment on the other hand. Furthermore, it is necessary to fine-tune the system in the real hardware by performing another few iterations of the genetic algorithm.

Although the implementation of the ANN has been done in software, due to choosing fixed-point arithmetics, the implementation is very resource-efficient and suited for the deployment on small embedded microcontrollers like the Atmel AVR series.

The ANN approach is also interesting when applied in real-time systems, where the maximum execution time of a task must be known in order to provide guarantees on the system's response time. Since a step of the ANN is very regular in terms of instruction sequence, the task's execution time is constant and the worst-case execution time of the tasks can be measured directly for typical microcontroller architectures without caches. However, for critical applications, it becomes difficult to guarantee for a particular behavior and to reason about this in safety-case issues.

Acknowledgments

This work was supported by the Austrian FWF project TTCAR under contract No. P18060-N04.

References

[1] J. Borenstein and Y. Koren. The vector field histogram - fast obstacle avoidance for mobile robots. *IEEE Journal of Robotics and Automation*, 7(3):278–288, June 1991.

[2] V. Braitenberg. *Vehicles: Experiments in Synthetic Psychology.* MIT Press, Cambridge, MA, USA, 1984.

[3] A. Elfes. A sonar-based mapping and navigation system. In *Proceedings of the IEEE International Conference on Robotics and Automation*, San Francisco, CA, USA, 1986.

[4] C. Emmanouilidis and A. Hunter. A comparison of crossover operators in neural network feature selection with multiobjective evolutionary algorithms. In *Proceedings of the GECCO-2000 Workshop on Evolutionary Computation in the Development of Artificial Neural Networks*, Las Vegas, NV, USA, July 2000.

[5] D. Floreano and F. Mondada. Automatic creation of an autonomous agent: Genetic evolution of a neural-network driven robot. In *From Animals to Animats 3: Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, pages 421–430, Brighton, United Kingdom, 1994.

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

[7] N. García-Pedrajas, D. Ortiz-Boyer, and C. Hervás-Martínez. An alternative approach for neural network evolution with a genetic algorithm: Crossover by combinatorial optimization. *Neural Networks*, 19(4):514–528, 2006.

[8] P. Husbands, I. Harvey, and D. Cliff. Analysing recurrent dynamical networks evolved for robot control. In *Proceedings of the 3rd IEEE International Conference on Artificial Neural Nets*. IEEE Press, 1993.

[9] G. Klingler, A. Kößler, and W. Elmenreich. The smart car - a distributed controlled autonomous robot. In *Proceedings of the Junior Scientist Conference 2006*, pages 33–34, Vienna, Austria, April 2006.

[10] H. Kraut. Signalverarbeitung mittels eines Neuronalen Netzwerkes für einen Smart Sensor. Bachelor's thesis, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, 2003.

[11] G. W. Lucas. *Rossum's Playhouse User's Guide for Version 0.60*. available at `http://rossum.sourceforge.net`, 2005.

[12] L. A. Meeden. An incremental approach to developing intelligent neural network controllers for robots. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 26(3):474–485, June 1996.

[13] S. Nolfi, D. Floreano, O. Miglino, and F. Mondada. How to evolve autonomous robots: Different approaches in evolutionary robotics. In *Proceedings of the International Conference Artificial Life IV*, pages 190–197, Cambridge, MA, USA, 1994. MIT Press.

[14] S. Nolfi, O. Miglino, and D. Parisi. Phenotypic plasticity in evolving neural networks: Evolving the control system for an autonomous agent. Technical Report PCIA-94-04, Institute of Psychology, C.N.R., Rome, Italy, 1994.

[15] A. Pfandler. Design and implementation of a generic framework for genetic optimization of neural networks. Bachelor's thesis, Vienna University of Technology, Vienna, Austria, 2007.

[16] A. Schörgendorfer. Extended confidence-weighted averaging in sensor fusion. Master's thesis, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, 2006.

[17] A. Schörgendorfer and W. Elmenreich. Extended confidence-weighted averaging in sensor fusion. In *Proceedings of the Junior Scientist Conference JSC'06*, pages 67–68, Vienna, Austria, April 2006.

[18] M. Sipper, Y. Azaria, A. Hauptman, and Y. Shichel. Designing an evolutionary strategizing machine for game playing and beyond. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 2006. to appear.

[19] M. D. Vose. *The Simple Genetic Algorithm: Foundations and Theory*. The MIT Press, Cambridge, MA, USA, 1999.