Automatically Designing Robot Controllers and Sensor Morphology with Genetic Programming

Bert Bonte, Bart Wyns

Automatically designing robot controllers and sensor morphology with genetic programming

Bert Bonte* and Bart Wyns

Ghent University, Deptartment of Electrical Energy, Systems and Automation,
Technologiepark 913, 9052 Zwijnaarde, Belgium
`Bart.Wyns@UGent.be,Bert.Bonte@gmail.com`
`http://www.UGent.be`

Abstract. Genetic programming provides an automated design strategy to evolve complex controllers based on evolution in nature. In this contribution we use genetic programming to automatically evolve efficient robot controllers for a corridor following task. Based on tests executed in a simulation environment we show that very robust and efficient controllers can be obtained. Also, we stress that it is important to provide sufficiently diverse fitness cases, offering a sound basis for learning more complex behaviour. The evolved controller is successfully applied to real environments as well. Finally, controller and sensor morphology are co-evolved, clearly resulting in an improved sensor configuration.

Key words: Genetic Programming, Evolutionary Robotics, Corridor Following, Sensor Morphology, EyeBot

1 Introduction

Robots are mainly used for relatively easy and repetitive tasks. Fully autonomous robots in realistic applications still are exceptions [16]. This is mainly caused by the highly complex design of such systems. More specifically, the development of robust controllers for real mobile robots is challenging.

In nature however, very powerful processes that are able to combine basic elements in a robust way resulting in very efficient and sometimes ingenious solutions exist. Using the concept of survival of the fittest, robot controllers and robot systems can be evolved just like organisms in nature. In this way, using natural processes, robust robot controllers can be automatically designed without complexity burden for human developers and even without programming.

The main objective of this contribution is developing robust controllers for corridor following using genetic programming [7]. The robot must navigate in a corridor system from start to end as efficiently as possible and without collisions. Unlike some other work in literature [17], we will not use a highly simplified simulation environment. The evolved mobile robot controllers must be robust enough to navigate successfully in corridor systems on which the robot was

* Presenting author

trained during the evolutionary process as well as new and unseen environments. Furthermore, the controller evolved in simulation must be transferable to the real robot preserving its behaviour learned in simulation. Finally, we also assess simultaneous evolution of controllers and sensor morphology using GP.

The remainder of this paper is as follows. Section 2 gives a brief overview of selected related work in literature. Then, Section 3 provides an introduction to GP, the standard parameters and the most important techniques used in the experiments. After that, in Section 4 the experimental setup is set out, including the simulation environment and the robot platform. Next, Section 5 discusses the evolution of robot controllers in simulation and the transfer to reality. Finally, Section 6 handles the combined evolution of controller and morphology.

2 Literature Overview

Evolutionary robotics in general [12] is a relative young discipline within the robotics research community, providing room for further improvement and advances. In this section we will give a brief literature overview closely related to the goals of this paper.

Most of the experiments are conducted in simplified simulation environments. The resulting controllers are limited to specific simulation parameters and conditions. Therefore, a lot of work in literature is devoted on the use of simulation environments or real robots [1, 4, 6, 11, 12, 17, 18]. Only using simple simulation environments without verifying on real robots afterwards is criticized a number of times [11, 12]. Indeed, there are significant differences between controllers performing well in simulation versus reality.

An example of a basic simulation environment is found in [8]. GP is used for evolving wall following-behaviour, which is part of many higher level robot skills. Successful programs were evolved, clearly exhibiting the desired behaviour. The simulation consisted of a grid environment were the “robot” is on one of the cells. The robot can directly move to neighbouring cells. Sensors simply indicate whether a certain cell contains an obstacle or not. Similar experiments provide some basic proof for using GP in robotics but their practical use is questionable.

In [17] a simplified but noisy simulation environment for evolving corridor following behaviour with steady-state GP was used. In this simulation, the robot moves forward with a constant speed, the controller only determines the direction. Moreover, the corridors are just small enough to let the robot pass, so turning in the wrong direction is impossible. Mostly short corridor systems were used. By adding noise to sensors and actuators Reynolds evolved more or less robust controllers.

In [3], a vision-based line-following controller was evolved in simulation by incrementally improving the visual model in the simulation. In simulation, successfull controllers using mostly basic functions were evolved. Mainly caused by some oversimplifications in the simulator, the authors were not able to successfully transfer this behaviour to the physical robot.

Some experiments were conducted directly on real robots. An example is in [14], where an obstacle avoidance controller for the *Khepera*-robot is evolved, again using steady-state GP. The population size was only 50 individuals. Two different training environments were tested. Using simple operations, in 200-300 generations the desired behaviour was achieved in both environments. The resulting program was robust, as it was successful in other environments as well.

3 Genetic Programming

Genetic programming uses a population of computer programs encoding a robot controller and was introduced by John Koza in the early nineties [7]. Each generation, only the best performing programs are selected for breading and have their offspring added to the population of the next generation. The selection of the best programs is based on the fitness function, which is defined to model how well the desired behaviour is matched. A detailed explanation on the fitness function used in this study is given in Section 3.2.

3.1 Standard Parameters

GP requires various parameters to be set. Our GP uses a population size equal to 500 and is allowed to run for 100 or 200 generations. Crossover (90%) and reproduction (10%) are used. As selection algorithm, tournament selection with seven individuals is used. The function set contains two functions: *IfLess* (arity 4), and *PROGN2* (arity 2), both well known. Terminals to move forward and backward over a distance of 10 cm are also included. Turn left and right makes the robot turn 15 degrees in place. Three infrared distance sensors are used: front, left and right, each perpendicular on the robot. Finally, three threshold values are available: low, medium and high, respectively 75, 150 and 300 mm.

3.2 Fitness Function

At each generation all individuals are tested in three environments and can perform 500 movements in each environment. The fitness function is averaged over all three tested environments and consists of three components:

- As a first component, the distance in bird's eye perspective the robot has covered so far is measured. This distance can be easily calculated by applying the rule of Pythagoras to the coordinates of the starting and ending position of the robot. This basic component mainly differentiates between controllers in initial generations.
- The second component punishes every collision detected using either sensor on each side of the robot. A collision occurs when the robot's sensor return a reading smaller than 50mm. The penalty consists of substracting a fixed value (3) to the number of movements so far.

– Finally, a bonus component is added when the robot reaches the end of the corridor system (robot's position equals top right corner of the simulation environment). This consists of a fixed part (3) and a variable part. The variable part is relative to the number of spare movements and thus rewards efficient controllers. The scalar used in this study equals to 4. This component mainly differentiates between better controllers in later generations.

While evaluating fitness on a single environment most likely leads to brittle strategies, averaging over multiple fitness cases results in more general solutions. Here, three different enviroments are used for evaluation. To select these three environments, we use a variant of the layered learning approach [5]. Two environments belong to one category, while the third belongs to a more difficult category. As the number of generations increases, the overall difficulty level of these three fitness cases also increases. However, changing the composition of the set of fitness cases at each generation has detrimental effects on the evolved controller quality. Therefore, some time (i.e. generations) is left for the evolutionary process to optimize the individuals towards the presented set of environments.

4 Experimental Setup

We use the *EyeBot*-platform for our experiments [1]. The evolutionary process is carried out entirely in the EyeSim, the simulation environment of the platform. Main advantage is that programs for the EyeSim can be transferred immediately for execution on the real EyeBot. Hence we realise a major speedup and cost reduction by evolving the controllers in simulation. To gain realism, we add our own additive noise component (Gaussian distributed noise), which can be manipulated easier. The standard deviation we use is 3 for sensor noise and 2 for motor noise. Combined with this noise, the simulation environment facilitates the evolution of controllers for real applications, which is impracticable in simplified and naive simulation environments.

The GP process was handled by ECJ [1], a research platform supporting multiple evolutionary algorithms in Java. ECJ constructs controllers of which the fitness is evaluated in the EyeSim and returned back to ECJ, which performs all evolutionary computations.

Since GP is a probabilistic method, we consider three different runs of each experiment. This relatively low number is justified because we are interested in the best controller, not some mean value. Moreover, when considering too many runs, the computational cost becomes too high.

5 Evolving Robot Controllers

As stated before in literature, noise can improve simulation results and lead to more robust controllers. Main argument is that with noise, evolution is no

[1] http://www.cs.gmu.edu/ eclab/projects/ecj/

longer able to exploit coincidences in fitness cases only valid in simulation and therefore leads to more robust controllers. Next to the noisy sensor values, noise is added to the steering mechanism as well. Two experiments with a population of 500 individuals were run, resulting in very high mean fitness values. The first one, using 100 generations lead to an extremely robust solution, performing well on five verification environments (not seen during training). Significant further improvement is possible since the entropy of 2.08 is extremely high. However, the second experiment using 200 generations, was slightly more efficient and fluent in turns. Though, this controller was not as robust as the first, presumably caused by premature convergence, in turn caused by too long training on preliminary categories. An example trail of both controllers is depicted in Figure 1. Further real life experiments will only use the first controller.

An interesting remark is that the underlying strategy of nearly all successful controllers of the experiments is wall following. Mostly, the left wall is followed, navigating to the end of the corridor system without turning and proceeding in the wrong direction. This general strategy is a nice example of evolutionary processes to come up with simple yet general and efficient solutions.

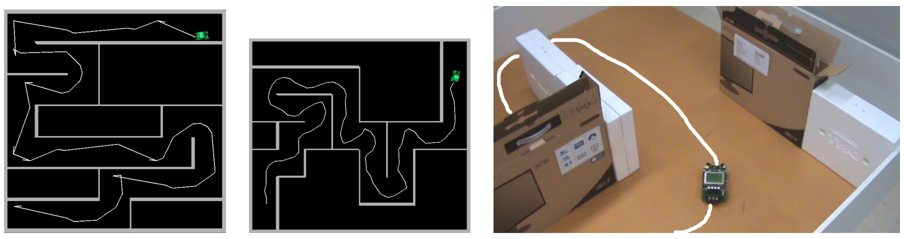

(a) (b) (c)

Fig. 1: Some example navigation trails. (a) The robot trail from the controller evolved in 100 generations. (b) The robot trail from the controller evolved in 200 generations. (c) The EyeBot with controller from (a) in a test environment. The white line denotes the robot trajectory.

When transferring the best controller to reality, performance slightly decreased. This was mainly caused by the fact that the real PSD sensors return increasing values when approaching a wall from a certain distance. After increasing the lowest threshold from 75 to 100 (the distance under which the sensor values become unreliable), this problem was solved. The robot was able to navigate efficiently through previously unseen environments. The robot successfully drives straight ahead, adjusts where necessary and most curves are taken smoothly. Nevertheless we noted slightly more collisions than in simulation. Figure 1(c) shows the real robot in a test environment. Remark that the left wall following is illustrated by omitting some right walls, yet resulting in a successful navigation.

6 Simultaneous Evolution of Controller and Sensor Morphology

6.1 Situation

In nature, both the controller and the body of an organism are evolved simultaneously. The analogy can be made to evolve controllers (software) and robot morphology (hardware), hopefully resulting in better suited robots and controllers for a given task. In this work, we will co-evolve the controller and sensor morphology [2, 17, 9, 10]. Again in simulation and with noise added to the sensor values. The standard simulation environment was extended to support the use of an arbitrary number of sensors.

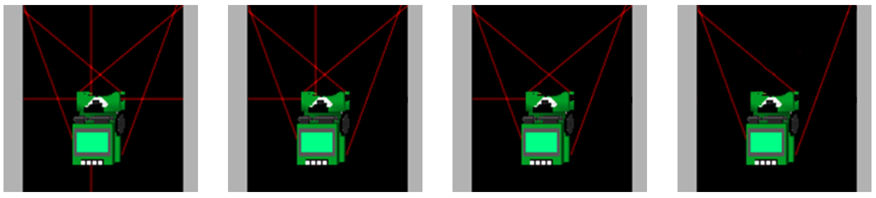
Because sensors are the main source of information on many robots, investigating their placement and quantity is an interesting task. Moreover, sensors can be moved easily on existing robots. However, the most important reason for examining sensor placement was found in preliminary experiments. Some – otherwise robust – controllers struggled with the detection of vertical walls. Since the frontal PSD-sensor measured parallel with the vertical wall, the latter was not detected and the robot repeatedly caused collisions. Therefore, the controller can be made more robust and general by better configuring sensor placement. Evolving sensor morphology can thus be advantageous.

6.2 Experimental Results

An experiment with 8 possible sensor locations, each addressed by a unique terminal leads to both a good controller and sensor configuration. In order to reach a minimal sensor suite, the fitness function awarded every successful controller according to the number of unused sensors. Note that the evolutionary process didn't reduce the number of sensors to evolve a simpler and more usable sensor morphology without this stimulus. The evolutionary process clearly omits the sensors one by one during evolution, starting with those that are least useful for the task. This is illustrated in Figure 2. In this case, these are the sensors that are placed orthogonal on the robot. Indeed, they only offer limited coverage (e.g. walls parallel with the sensor placement). Therefore the evolutionary process selects almost exclusively non perpendicular sensors, especially at the front of the robot. This is an important design issue that was not taken in account when designing the real robot, which only has perpendicular sensors.

A similar experiment with 35 sensors resulted in a good sensor configuration as well, again with non perpendicular sensors on the front. However, the resulting controller was not robust enough. Here we reached a complexity barrier for the evolutionary process; too many terminals lead to an explosion of the search space resulting in a problem that is too hard to solve with a standard general purpose PC in an acceptable time span. This is a concern that is shared by other authors in the field as well [15].

Another approach with one generic function for all possible sensor positions accepting three parameters (x and y position and angle on the robot) defining



(a) Generation 0 (b) Generation 15 (c) Generation 25 (d) Generation 30

Fig. 2: The sensor usage during the evolutionary process. Sensors with limited ranges in terms of the measured surface are omitted during initial generations.

the sensor placement was investigated as well. Again we found that the fitness function must reward the limited use of sensors, otherwise too many sensors are used. This results in unnecessary complex controllers. However, more research is needed for quantifying the difference in result and computation time between a number of strictly defined terminals or a more generic function which is more flexible.

7 Conclusions

We demonstrated that, even with a basic PC and limited computation time, GP is able to evolve controllers for corridor following in a simulation environment by using a gradual form of layered learning. Moreover, this controller was transferred successfully to reality. Secondly, we also co-evolved controller and sensor morphology of the robot. This resulted in a significantly improved sensor configuration.

References

1. Bräunl, T. (2006). *Embedded Robotics: Mobile Robot Design and Applications With Embedded Systems*. Springer-Verlag, 2nd edition edition.
2. Buason, G., Bergfeldt, N., and Ziemke, T. (2005). Brains, bodies, and beyond: Competitive co-evolution of robot controllers, morphologies and environments. *Genetic Programming and Evolvable Machines*, 6(1):25–51.
3. Dupuis, J. and Parizeau, M. (2006). Evolving a Vision-Based Line-Following Robot Controller. In *Proceedings of the The 3rd Canadian Conference on Computer and Robot Vision*, page 75. IEEE Computer Society Washington, DC, USA.
4. Goosen, T., Brule, R., Janssen, J., and Haselager, W. (2007). Interleaving Simulated and Physical Environments Improves Evolution of Robot Control Structures. In Dastani, M. and de Jong, E., editors, *BNAIC 2007: Proceedings of the 19th Belgium-Netherlands Artificial Intelligence Conference, November 5-6, Utrecht, the Netherlands*, BNAIC 19, pages 135–142. Utrecht University.
5. Gustafson, S. and Hsu, W. (2001). Layered Learning in Genetic Programming for a Cooperative Robot Soccer Problem. *Lecture Notes in Computer Science*, 2038:291–301.

6. Jakobi, N., Husbands, P., and Harvey, I. (1995). Noise and the reality gap: The use of simulation in evolutionary robotics. *Lecture Notes in Computer Science*, 929:704–720.
7. Koza, J. (1992). *Genetic Programming: On the programming of computers by Means of natural selection*. MIT Press, Cambridge, Massachusetts.
8. Lazarus, C. and Hu, H. (2001). Using Genetic Programming to Evolve Robot Behaviours. In *Proceedings of the 3rd British Conference on Autonomous Mobile Robotics & Autonomous Systems*.
9. Lee, W. (2000). Evolving Autonomous Robot: From Controller to Morphology. *IEICE Transactions on Information and Systems*, 83(2):200–210.
10. Lund, H., Hallam, J., and Lee, W. (1997). Evolving robot morphology. In *IEEE International Conference on Evolutionary Computation*, pages 197–202.
11. Nelson, A., Grant, E., Galeotti, J., and Rhody, S. (2004). Maze exploration behaviors using an integrated evolutionary robotics environment. *Robotics and Autonomous Systems*, 46(3):159–173.
12. Nolfi, S. and Floreano, D. (2000). *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*. MIT Press Cambridge.
13. Nordin, P. and Banzhaf, W. (1995a). Complexity compression and evolution. In Eshelman, L., editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 240–245. Morgan Kauffman.
14. Nordin, P. and Banzhaf, W. (1995b). Genetic programming controlling a miniature robot. In *Working Notes for the AAAI Symposium on Genetic Programming*, pages 61–67.
15. Ok, S., Miyashita, K., and Nishihara, S. (2000). Improving performance of gp by adaptive terminal selection. In *PRICAI 2000 Topics in Artificial Intelligence: 6th Pacific Rim International Conference on Artificial Intelligence*, volume 1886, pages 435–445. Springer.
16. Pollack, J., Lipson, H., Ficici, S., Funes, P., Hornby, G., and Watson, R. (2000). Evolutionary techniques in physical robotics. In *Evolvable Systems: From Biology to Hardware: Third International Conference, ICES 2000, Edinburgh, Scotland, UK, April 17-19, 2000: Proceedings*. Springer.
17. Reynolds, C. (1994). Evolution of corridor following behavior in a noisy world. In Cliff, D., Husbands, P., Meyer, J.-A., and Wilson, S., editors, *From Animals to Animats 3: Proceedings of the third International Conference on Simulation of Adaptive Behavior*, pages 402–410. MIT Press.
18. Urzelai, J. and Floreano, D. (2000). Evolutionary Robotics: Coping with Environmental Change. In *Proceedings of the Genetic and Evolutionary Computation Conference*.