Evolving a Vision-Based Line-Following Robot Controller

Jean-François Dupuis and Marc Parizeau
Département de génie électrique et de génie informatique,
Université Laval, Québec (QC), Canada, G1K 7P4.
jfdupuis@gel.ulaval.ca, parizeau@gel.ulaval.ca

Abstract

This paper presents an original framework for evolving a vision-based mobile robot controller using genetic programming. This framework is built on the Open BEAGLE framework for the evolutionary computations, and on OpenGL for simulating the visual environment of a physical mobile robot. The feasibility of this framework is demonstrated through a simple, yet non-trivial, line following problem.

1 Introduction

Evolutionary Robotics (ER) is a research field related to behavioral robotics [15, 16]. In ER, evolutionary computation methods are applied to automate the development of autonomous robot controllers. Evolutionary computations use artificial evolution methods on a population of initial random individuals that evolve over time within a given environment. These individuals represent specific solutions to the problem at hand. They are tested and ranked according to the quality of their task accomplishment, as defined by an objective function. Higher reproduction probability are given to the best performing individuals. The reproduction process is combined with genetic operators, such as crossover and mutation, to engender new generations of individuals. This process is continued, generation by generation, until some stopping criterion is reached [1].

ER is commonly used in the development of robot controllers for maze navigation and obstacle avoidance tasks using proximity sensors [9, 15, 12]. Others have used evolutionary processes to develop walking gaits controllers [4, 7]. Likewise, colony of robots have also been evolved to find emergent collaborative behaviors [13]. Moreover, the robot platform itself can be been evolved in symbioses with its controller [10].

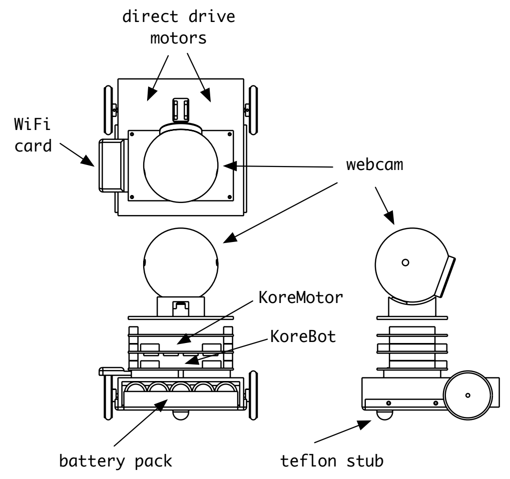
However, the use of visual input in ER has been seen only in a few limited instances. Harvey first evolved a suspended robot in 1994 within an artificial environment [5]. He used a sub-sampled signal from a 64x64 monochrome CCD camera as visual input. The genetically evolved algorithm was in charge of determining the subsampling strategy. The robot was continuously connected to an offboard computer. In another project, Nolfi and Floreano used coevolution to develop strategies in a prey-predator context [14]. In this experiment, a linear gray scale visual input was used. This time, all of the computations were done onboard. More recently, a linear visual input was also used in a vision-based flying robot project [17].

A high resolution camera was seen in [11], where a 640x480 USB color camera connected to a PC/104 computer was used, doing all computations onboard. However, as input to the controller, the color space of the image was first reduced to four levels: red, green, blue, or other.

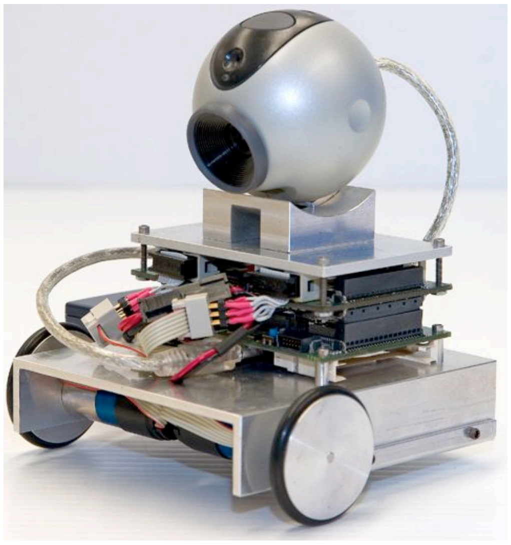
All of these previous work have used visual information as input for a genetically evolved neural network controller. However, they only used part of the information available by either subsampling the signal or by using very low resolution sensing devices. Beside, all of the evolutionary processes were conducted on the robot, whitout the use of simulation.

The goal of the present project is to develop a vision-based controller for an autonomous mobile robot using genetic programming [1, 8]. In this paper, we describe the actual robotic platform that was developed for this project, including a software simulator for this platform, and we report on some preliminary results that were obtained for a relatively simple, but not trivial, line following task. The goal here, is not to develop a visual servoing algorithm [6] per se, but to obtain emerging visually driven behaviors for a mobile robot wandering in an unconstrained environnement using evolutionnary computation.

The outline of the paper is as follows. The next section gives an overview of the physical robot, of the robot environment, and of the robot simulator. Then, Section 3 presents the evolutionary setup that was used to conduct the experiments for which results are analysed in Section 4. Finally, Section 5 concludes the paper with perspectives and future work.



(a) design views



(b) photograph

Figure 1. Physical robot: (a) design views; (b) photograph.

2 Robotic platform

A custom 110 mm long by 125 mm wide two motorized wheel robot was developed for this project (see Figure 1). The robot is equipped with a KoreBot card[1], which is a miniature Linux-based single-board computer, with an Intel XScale 400 MHz PXA-255 processor, 64 MB of RAM, 32 MB of Flash memory, USB port, and PCMCIA card slot. This board is combined with the KoreMotor card, which is a motor controller card implementing PID control to DC motors.

Video data acquisition is through a USB video camera connected to the KoreBot. The camera is able to deliver a compressed image resolution of 640x480 pixels at 30 FPS. However, because of current software limitations, the camera is used in it uncompressed video format, namely YUV420, of 160x120 pixels at 30 FPS. Even though computations are generally done onboard, the robot can communicate with a remote host through an embedded WiFi link. This link is used for externally supervising the robot, and to download/upload code and data.

[1]http://www.k-team.com

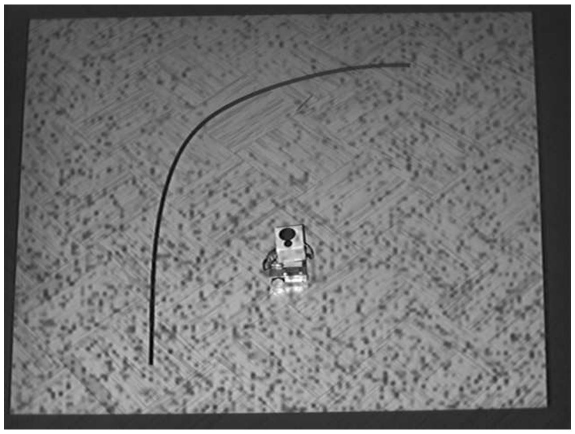
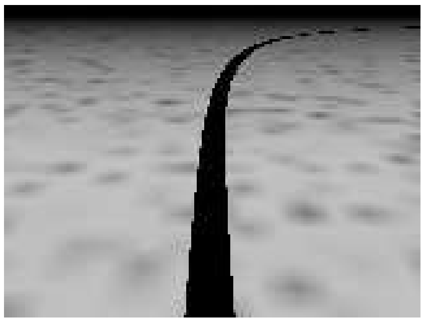


Figure 2. The robot in his environment in a line following task viewed from the supervisor.

2.1 Dynamic environment

The robot environment is artificially controlled using an LCD projector that projects images on the floor. This setup is interesting because it enables instant dynamical reconfiguration of the environment without any human intervention. For the case of the line following problem, the line can be changed easily in between evaluation runs, and obstacles or other features can be added dynamically to the scene. Figure 2 shows the robot in action. The image background is a mix of the natural texture from the floor tiles in our lab, and of random dot texture generated through the projector.

In addition to this projector, the environment also contains a supervisor camera which looks down at the scene (camera axis is parallel to projector axis), tracking and monitoring the robot movements. This information is used for the fitness evaluation of the genetic programs that are ran on the physical robot, and for remote positioning of the robot at the beginning of each evaluation run. It can also be used as an external agent, like a prey in a prey-predator context. This dynamic environment thus offers a great versatility on the kind of feasible tasks.


(a) Simulated image

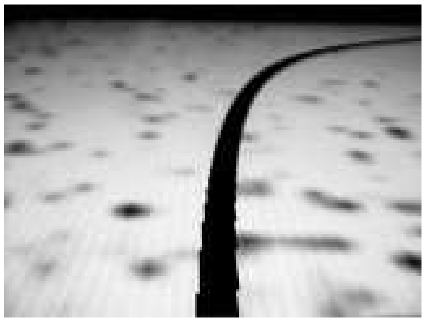

(b) Robot camera

Figure 3. Image taken from the robot view.

2.2 Robot simulator

The main problem with evolutionary computations is fitness evaluation time. Using a single physical robot to evaluate the fitness of a population of n individual controllers requires n sequential evaluation runs. To enable parallel fitness evaluation requires either an army of robots, or a robot simulator and a cluster of computers running virtual robots in a virtual environment.

A robot simulator was thus created using an OpenGL engine to generate realistic visual information for a virtual robot. The scene model can be rendered with good fidelity as can be seen by comparing the images of figure 3. The fact that the real scene is already artificially manufactured (except for the texture of the floor tiles) greatly facilitates the development of a good visual model.

The computation time of the genetic programming primitives used by the controller is another important parameter of the simulation. Obviously, the refresh rate of the motor commands has a great influence on the robot's behavior. If the refresh rate is too slow, then the robot may not be able to react efficiently to its sensory input. In the particular case of the line following problem, the robot may simply speed along and miss a curve in the line because it was never seen. Thus the ratio of processing speeds between the physical and simulated robots must be taken into account carefully.

3 Experimental setup

For the reported experiments, the robot controller was trained for a line following task. The scene is a 5m $\times$ 5m clear gray plane surface without any walls, overlaid with random darker gray spots. Outside this surface, there is a uniform dark background. A black NURBS curve is drawn around the center region of this scene (Figure 2). This curve is logically decomposed into small segments, and the task is to visit a maximum number of these segments in a specified amount of time. Here, each line was decomposed into 200 fragments of equal length. As the lenght of the lines used is not kept constant, the segment length will vary for each line.

3.1 Fitness evalutation

When evaluating the fitness of evolved controllers, each of them is tested three times at each generation, for three different lines and initial position. The fitness is then the mean of these three evaluation runs. In this way, lucky individuals are less likely to have a good fitness, while the really good ones should be able to come through in successive runs. By changing the line, we make sure that the controllers do not get over-trained for a particular line.

To facilitate the initial development of individuals in the first generations of evolution, it is important to increase the contrast between the line and the background as much as possible. The visual model was thus incrementally made more realistic in three steps. In a first step, during the first generations, the background color is set to medium gray (instead of black) so that the horizon cannot be confused with the pursued line. Later on, in a second step, the background is turned to black just like in the physical scene. Finally, in a third step, the visual model is completed by a linear fog rendering which blends the scene pixels with the background color, proportionally to the depth of the scene. The effect of this fog rendering is shown in Figure 4.

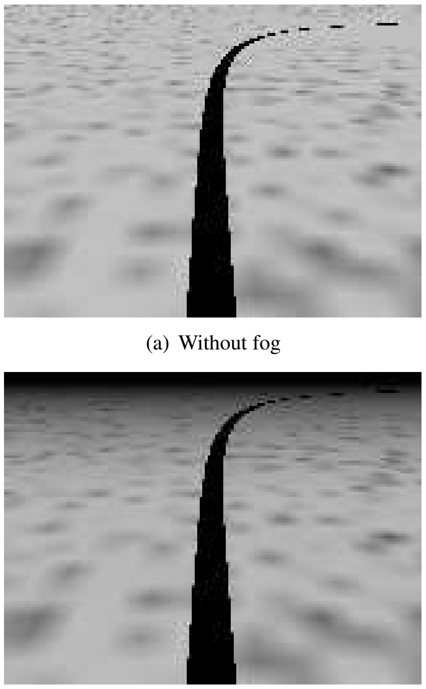
3.2 Fitness function

The objective function $\phi(i)$ used to estimate the fitness of an individual run i is simply the fraction of line segments that were visited during this run:

$$\phi(i) = \frac{1}{S_i}\sum_{j=1}^{S_i} s_j, \qquad (1)$$

where S_i is the total number of segments for the line of run i, and s_j is either 1 if segment j was visited by the robot, or 0 otherwise. The final fitness Φ is the average of q distinct runs:

$$\Phi = \frac{1}{q}\sum_{i=1}^{q} \phi_i \qquad (2)$$


(a) Without fog

(b) With fog

Figure 4. Image taken from the robot view with and without fog.

A line segment is said to be visited if, for any of the time steps, the nearest point on the line to the robot position is situated on that segment and that the distance from the robot to the line is smaller than a specified threshold. For our experiments, we have used a fixed threshold of 10cm.

Some other parameters to this simple fitness calculation were also tried. For example, the collinearity of the robot speed vector with regard to the line slope, or the time passed away from the line. But results were not conclusive. A multiobjective optimization could possibly generate better results, but more work is needed in order to define a more sophisticated fitness function.

3.3 Genetic Programming

In Genetic Programming (GP) individuals are represented by computer programs that usually take the form of rooted trees of primitive operations [1, 8]. Initially, random trees are generated, and genetic operations involve alteration of subtrees. Crossover operations are usually implemented by crossing two subtrees taken from two individuals. One kind of mutation consists in replacing a subtree by another randomly created.

To setup a GP evolution, the first step is to define an adequate set of primitives operations, that will be used by the

Table 1. Summary of GP parameter settings

Objective	Visual line following by visiting the maximun number of line segments
Terminal set	Input image and ephemerals
Function set	ADD, SUB, MUL, DIV, IfThenElse, MAX, MIN, MEAN, MEDIAN, STDV, CONV, THRESH, CoGH, CoGV, MMEAN, EXTRACT
Crossover probalibility	90%
Mutation probability :	
Standard	5%
Shrink	5%
Swap	5%
Selection	Decimate, oversize ratio 7
Population size	100
Testing time	20 seconds
Time factor	5
Incrementation steps :	
Background	15 generations
Fog	30 generations

GP process as tree nodes. The primitives that were used in this work are summarized in Table 1. The set of terminals (primitives without arguments) consists of the camera image, on the one hand, and of ephemeral matrices and scalars, on the other hand. Ephemeral primitives are randomly generated constants that are set during individual inception. In order to deal with different types of data that are processed in the GP trees, such as matrix versus scalar, constrained GP is used [8]. This ensures data compatibility between primitives.

In the set of primitives, arithmetic operators such as addition (ADD), subtraction (SUB), multiplication (MUL) and division (DIV), are defined. When these operators are applied to a mixture of scalar and matrix, the operation is element wise. A special transformation is applied when the size of the matrix arguments are size incompatible. This transformation consists in first aligning their centers, and then extracting their overlapping area.

Filtering primitives are also defined for maximum (MAX), minimum (MIN), mean (MEAN), median (MEDIAN), standard deviation (STDV) and convolution (CONV) masking operations. These primitives operate on two operands, the first is interpreted as the image matrix, and the other specifies the mask size for the operation, except for primitive CONV which receives the mask itself as the second argument. The mask size argument is interpreted either as 3×3, 5×5, or 7×7.

Other primitives allow some simple computations such as horizontal or vertical center of gravity, binary threshold, or matrix mean (MMEAN). Also, an extractor operator (EXTRACT) can extract a sub-matrix. It should be noted that center of gravity and matrix mean operations are the only ones that receive a matrix argument and that return a scalar value.

The controller output is generated at the root node of the program tree. This special root node receives two scalar values as arguments. The first is interpreted as an angular velocity for the robot, ω, and the other as its forward speed, V. The tangential wheel speeds are then determined using the kinematic model of the robot, which takes into account its wheelbase, B.

$$v_l = V - \frac{\omega B}{2}, \qquad v_r = V + \frac{\omega B}{2} \tag{3}$$

where v_l and v_r are respectively the left and right tangential wheel velocities. This approach was found to be more succesful than interpreting the values received at the root node directly as motors speeds.

From the command generation algorithm, it is apparent that motor commands are refreshed after every execution of the program tree. Because the refresh rate of the commands are inversely proportional to the tree size, which influences the performance of controllers, this constitutes an implicit parsimony pressure on the size of the individuals that counters the natural bloating tendency of GP programs during evolution.

The evolutionary algorithm described above was implemented using the Open BEAGLE C++ framework[2] for evolutionary computations [3], and its companion Distributed BEAGLE[3] that creates a master-slave architecture to run the evolution on a cluster of computers [2].

4 Results

Successful behaviors were obtained in simulation using the parameters listed in table 1. The selection algorithm used was the decimation operator, which kept the n best individuals of the population. Here, n is the population size. In conjonction with the decimation operator, the oversize operator was used to generate the *ratio* $*$ *population size* children individuals from a population of n parents.

Figure 5 summarizes the main statistics of the evolution: fitness, tree depth, and tree size. The graph of fitness vs generation shows that the evolution mostly stalls after 25 generations. The average population fitness is about 35%, while the best individual reached a little over 70%. Nevertheless, these are good results considering the severe time limit constraint that was imposed on the robot for completing his run. In 20 seconds, most robot controllers are not able to visit the

[2]http://beagle.gel.ulaval.ca

[3]http://beagle.gel.ulaval.ca/distributed

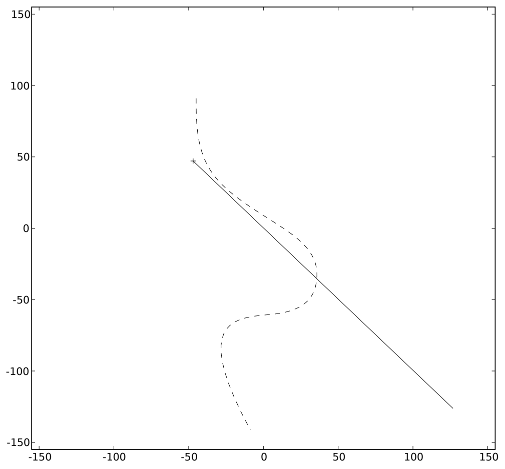


Figure 7. Wrong behavior having good fitness value. In this case, the fitness associated to this individual is 0.40.

complete line, even if they are initially positioned at one of its end-points. Moreover, the robots were positioned mostly around the center of the scene by the fitness evaluation process, even though their exact position and orientation was determined randomly. The graphs of the tree depth and tree size give some hint at the bloat phenomenon that was mentioned in the previous section. The two graphs show that the evolved programs tend to grow in complexity through time, despite the parsimony pressure imposed by the time limit.

Figure 6 shows the behavior of some of the successful controllers. It also illustrates the different paths that were used to test the evolved controllers.

One of the problem with our objective function is that it does not discriminate well enough between good controllers that are able to follow lines, and trivial ones that adopt strategies like always going straight. One such example is given in Figure 7. where the corresponding controller received a fitness of 40%, above the population average. More work is needed to refine the objective function so that it deals adequately with such undesirable behavior.

Another manifestation of the same problem is that even the good controllers sometimes have difficulty in locating the line quickly and, hence, are not able to demonstrate their abilities for each and every run. Thus a bad run may also bring them toward the average population fitness.

For this reason, a new evaluation procedure is currently being investigated. Rather than having a predefined amount of time for each evaluation, the individual controller will receive additionnal time for each line segment visited. This can be seen as food placed along the line to feed the robot with additional energy to continue. This should help to solve the problem of good controllers that are unable to fin-

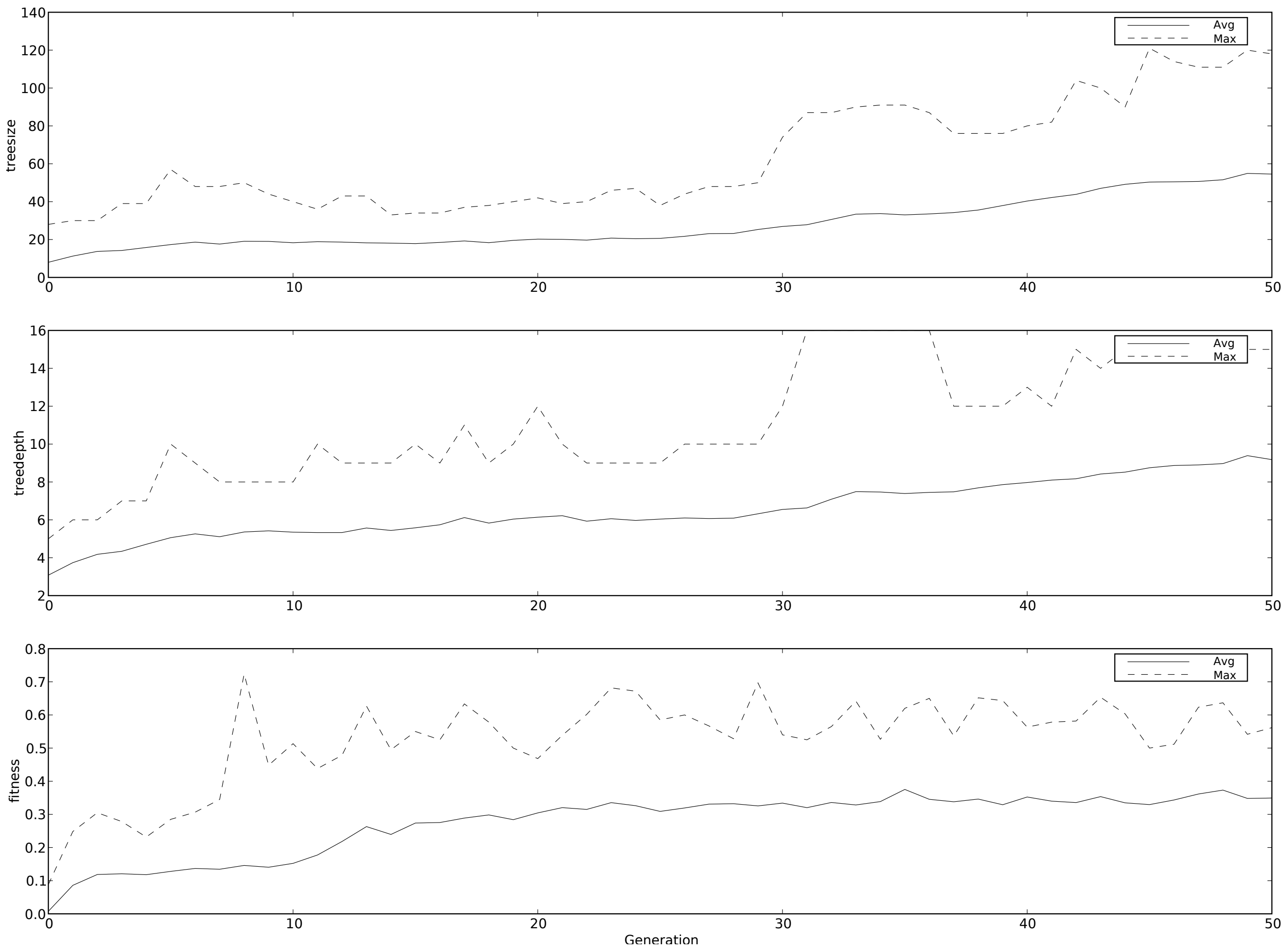


Figure 5. Progression of tree size and depth and the fitness over all the generations.

ish their work.

In the present research state, behaviors obtained in simulation were unsuccessfully transfered to the physical robot. The reason is that the simulator still presents some over-simplifications. In fact, the computation time model should be more elaborate and a dynamic model of the robot should be incorporated into the simulator to better reflect the exact response of the robot to the generated commands. These simplifications allowed us to rapidly generate simulation results that now justify further development effort.

5 Conclusion

In this paper, an innovative dynamic environment for vision-based evolutionary robotics research was described. The autonomous mobile robot rely on gray scale video data to perform the easy but non trivial task of line following. Genetic programming was used to successfully evolve sought behaviors in simulation. The required simulation time was greatly reduced by distributing the evaluation process over a cluster of 24 computers.

Although the results presented here were only based on simulation, ongoing work on the refinement of the robot model should lead to a successful transfer of evolved controllers to the physical robot. Besides, a better fitness function that would clearly dissociate the sought behavior should greatly help the convergence of the evolutionary process. Finally, we are also investigating the addition of state primitives in our genetic programming toolbox, in order to enable the exploitation of past experiences.

References

[1] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone. *Genetic Programming – An Introduction; On the Automatic Evolution of Computer Programs and its Applications*. Morgan Kaufmann, dpunkt.verlag, 1998.

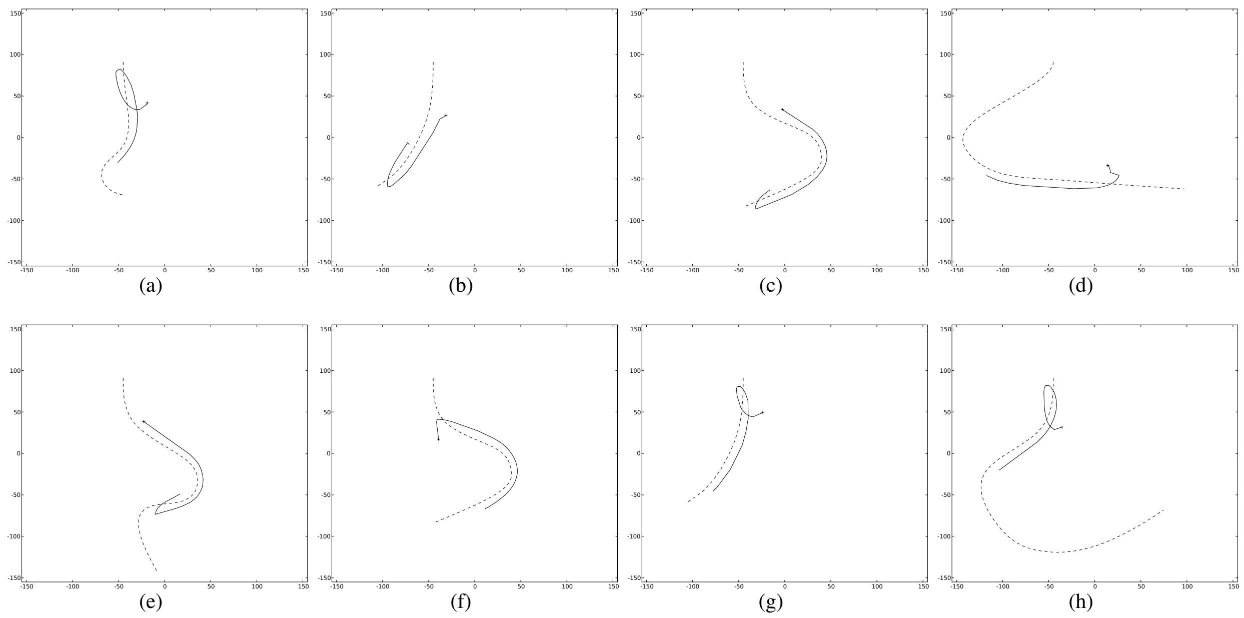


Figure 6. Path traveled by individuals of the last generation. A small marker at the beginning of the line indicates the robot's initial position.

[2] M. Dubreuil, C. Gagné, and M. Parizeau. Analysis of a master-slave architecture for distributed evolutionary computations. *IEEE Transactions on Systems, Man, and Cybernetics*, 36:229–235, February 2006.

[3] C. Gagné and M. Parizeau. Genericity in evolutionary computation software tools: Principles and case study. *International Journal on Artificial Intelligence Tools*, in-press 2006.

[4] T. Gomi and K. Ide. Evolution of gaits of a legged robot. *Special Issue on Learning in Autonomous Robots, Autonomous Robots Journal*, 1997.

[5] I. Harvey, P. Husbands, and D. Cliff. Seeing the light: artificial evolution, real vision. In *SAB94: Proceedings of the third international conference on Simulation of adaptive behavior : from animals to animats 3*, pages 392–401, Cambridge, MA, USA, 1994. MIT Press.

[6] K. Hashimoto. A review on vision-based control of robot manipulators. *Advanced Robotics*, 17(10):969–991, 2003.

[7] N. Jacobi. Running across the reality gap: Octopod locomotion evolved in a minimal simulation. In *Proceedings of the First European Workshop on Evolutionary Robotics*, pages 39–58, London, UK, 1998. Springer-Verlag.

[8] J. R. Koza. *Genetic programming: on the programming of computers by means of natural selection*. MIT Press, Cambridge, MA, USA, 1992.

[9] W.-P. Lee. Evolving complex robot behaviors. *Information Sciences*, 121:1–25, December 1999.

[10] H. Lipson and J. B. Pollack. Automatic design and manufacture of robotic lifeforms. *Nature*, 406(6799):974–978, August 2000.

[11] A. L. Nelson, E. Grant, G. Barlow, and T. Henderson. A colony of robots using vision sensing and evolved neural controllers. In *IEEE/RSJ International Conference On Intelligent Robots And Systems*, volume 3, pages 2273–2278, 2003.

[12] A. L. Nelson, E. Grant, J. M. Galeotti, and S. Rhody. Maze exploration behaviors using an integrated evolutionary robotics environment. *Robotics and Autonomous Systems*, 46:159–173, March 2004.

[13] A. L. Nelson, E. Grant, and T. C. Henderson. Evolution of neural controllers for competitive game playing with teams of mobile robots. *Robotics and Autonomous Systems*, 46:135–150, March 2004.

[14] S. Nolfi and D. Floreano. Coevolving predator and prey robots: Do "arms races" arise in artificial evolution? *Artif. Life*, 4(4):311–335, 1998.

[15] S. Nolfi and D. Floreano. *Evolutionary Robotics. The Biology, Intelligence, and Technology of Self-organizing Machines*. MIT Press, Cambridge, MA, 2001.

[16] J. Walker, S. Garrett, and M. Wilson. Evolving controllers for real robots: A survey of the literature. *Adaptive Behavior*, 11(3):179–203, 2003.

[17] J. Zufferey, D. Floreano, M. van Leeuwen, and T. Merenda. Evolving vision-based flying robots. In Blthoff, Lee, Poggio, and Wallraven, editors, *Proceedings of the 2nd International Workshop on Biologically Motivated Computer Vision (LNCS)*, pages 592–600. Springer-Verlag, 2002.