Evolution of a Tactile Wall-Following Behavior in Real Time

Frank Hoffmann[1] and Juan C. S. Zagal Montealegre[1]

Royal Institute of Technology, NADA/CVAP, 10044 Stockholm, Sweden
{hoffmann,juan}@nada.kth.se,
WWW home page: http://www.nada.kth.se/~hoffmann

Abstract. This paper describes the evolution of a simple, reactive wall-following behavior. The robot perceives its environment by means of tactile sensors and is supposed to travel along a looping maze as smoothly as possible. A genetic algorithm learns the optimal mapping from sensory information to motor actions. Candidate behaviors are executed on a LEGO Mindstorm robot and the distance covered during the trial serves as a fitness measure.
A model of the fitness function is build from a small number of behavior evaluations on the real robot. The genetic algorithm utilizes this model in the selection phase to predict the fitness of a candidate behavior. Every few generations the entire population is updated according to the true fitness evaluated on the real robot, and the new samples are used to update the fitness landscape model.

Keywords: evolutionary robotics, genetic algorithms, behavior based robotics, locally weighted regression

1 Introduction

Evolutionary robotics is a design paradigm to adapt robotic behaviors by means of simulated evolution[9]. The basic idea it to use evolutionary computation in order to automatically synthesizes robot controllers that exhibit a useful behavior perform in a complex environments. The evolutionary algorithm searches through space of possible mappings from sensory perceptions to control actions. A population of candidate controllers is processed from one generation to the next by means of selection, recombination and mutation. A scalar fitness function describes the performance of the controller with respect to the task assigned to the robot.

In practice, the evaluation of the controller often takes place in a simulator of the robot and the environment [7]. The underlying assumption of this approach is that the simulation is authentic enough such that controllers once transfered to the real robot exhibit a similar performance. Otherwise, it might that happen that the evolutionary algorithm exploits peculiarities of the simulator that possess no actual counterpart in the real world. JAKOBI proposed a a framework

called "minimal simulation" for the construction of fast-running robotic simulators [8]. The basic idea is to model only those aspects that are highly relevant to the behavior in mind and thereby to assure that the evolved controllers are more robust against uncertainty and incomplete information inherent to real world situations.

The alternative approach is to evolve behaviors directly on the robot itself as the world is its best own model [4]. Numerous authors report experiments in which the evolutionary learning takes place on the real robot [5, 3].

2 Genetic Algorithm

Evolutionary algorithms provide a universal optimization technique that mimics the type of genetic adaptation that occurs in natural evolution [6]. Unlike specialized methods designed for particular types of optimization tasks, they require no particular knowledge about the problem structure other than the objective function itself.

A population of competing candidate solutions evolves over time by means of genetic operators such as mutation, recombination and selection. Genetic algorithms operate on binary strings $\boldsymbol{s} = \{s_1, \ldots, s_N\}$ that are mapped into potential solutions to the optimization problem. A scalar objective function $f(\{s_1, \ldots, s_N\})$ evaluates the quality of a candidate string $\{s_1, \ldots, s_N\}$. The role of selection is to exploit good candidate solutions by allowing those strings that achieve a high fitness to reproduce offspring. Crossover cuts and splices two parent strings in order to generate new variants. Mutation randomly alters individual and thereby maintains the genetic diversity in the population. The interplay of exploiting good solutions via selection and exploring the search space in form of crossover and mutation constitutes the fundamental theme in evolutionary optimization. The quality of solutions improves gradually as a result of this basic cycle of selection, reproduction, recombination and mutation.

3 Learning a Fitness Model

In an evolutionary algorithm evaluating the fitness of a candidate solution is computationally expensive compared to which the computational effort associated with selection, crossover and mutation can be usually neglected. This observation becomes particularly relevant when evolving robotic behaviors, as evaluating the performance on the real robot is extremely time consuming. Therefore, a scheme that is able to reduce the number of true fitness evaluations required by the evolutionary algorithm carries a substantial benefit.

In [10], a model-based learning strategy is proposed. Samples of the true fitness function are used to learn a fitness landscape model that is used by the genetic algorithm during selection. After a few generations of model-based evolution the true fitness of individuals is evaluated and the fitness model is refined by means of the newly collected samples.

Locally weighted regression [1] forms an approximation of the fitness function $\hat{f}(\boldsymbol{s})$ based on samples $\{\boldsymbol{s}, f\}$ of the underlying true fitness function $f(\boldsymbol{s})$. Given a new query string $\boldsymbol{s}_q$ a local approximation

$$\hat{f}(\boldsymbol{s}_q) = \frac{\sum_i f^K(d(\boldsymbol{s}^q, \boldsymbol{s}^i))}{\sum_i K(d(\boldsymbol{s}^q, \boldsymbol{s}^i))} \tag{1}$$

is computed by distance weighted regression over the training examples. The kernel function $K(d(\boldsymbol{s}^q, \boldsymbol{s}^i))$ decreases as the distance $d(\boldsymbol{s}^q, \boldsymbol{s}^i)$ between the query point $\boldsymbol{s}^q$ and the training sample $\boldsymbol{s}^i$ increases. The distance

$$d(\boldsymbol{s}^q, \boldsymbol{s}^i) = \sum_{n=1}^{N} 1 - \delta_{s_n{}^q, s_n{}^i} \tag{2}$$

between two points is computed as the Hamming distance of the binary strings. The kernel function

$$K(d) = \frac{1}{1 + d^2} \tag{3}$$

has its maximum at zero distance and goes to zero for increasing Hamming distance.

A fitness model build from a small number samples during the early stages of the evolutionary algorithm only provides a rough approximation of the underlying fitness landscape. Therefore, a sampling strategy is needed which determines whether the model is used to predict the fitness of a new individual, or if it is evaluated based on the true fitness function, which value is more accurate but also more costly to obtain. In general any model learned from a set of training data should minimize the expected modeling error

$$E = \int (\hat{f}(x) - f(x))^2 p(x) dx \tag{4}$$

over the distribution $p(x)$ of future samples. If the learner is able to select the training examples $\{f(x_i), x_i\}$ itself, as it is the case of modeling a fitness function, new samples should be drawn at those regions in which the modeling error is maximal. The modeling error can be estimated by means of cross-validation over the current set of training examples.

In a genetic algorithm selection favors individuals with larger fitness such that the population ultimately converges to the peaks of the fitness landscape. Therefore, the fitness model should be more accurate in regions of high expected fitness compared to those of low fitness. The objective of a fitness model for the genetic algorithm is to minimize the modeling error over the distribution of individuals in future generations. Accordingly, the sampling strategy draws samples of the true fitness function for individuals x_i that have a high expected fitness $\hat{f}(x_i)$ and a large expected modeling error $< (\hat{f}(x_i) - f(x_i))^2 >$. A simple strategy is to evaluate the k individuals with the largest sum of expected fitness and modeling error on the true fitness function, and to use the approximate model to determine the fitness of the remaining individuals.

4 Robotic Architecture

The LEGO Mindstorms set allows non-expert users to build and program their own robotic systems. The Robotic Controller X (RCX) is a a programmable micro-controller packaged in a palm-size LEGO brick. The RCX can receive input from up to three touch, light or rotation sensors and is able to control up to three outputs connected to electric motors. A resistor network mimics a simple digital to analog converter which allows it to connect three touch sensors to a single RCX input. This three touch sensor expander allows the RCX to monitor six touch sensors and a rotation sensor.

The RCX communicates with a remote PC via an infrared serial link. This feature is used in our experiments to down-load the behavior parameters from the PC to the RCX and to up-load the scalar fitness value from the RCX to the PC upon execution of a behavior.

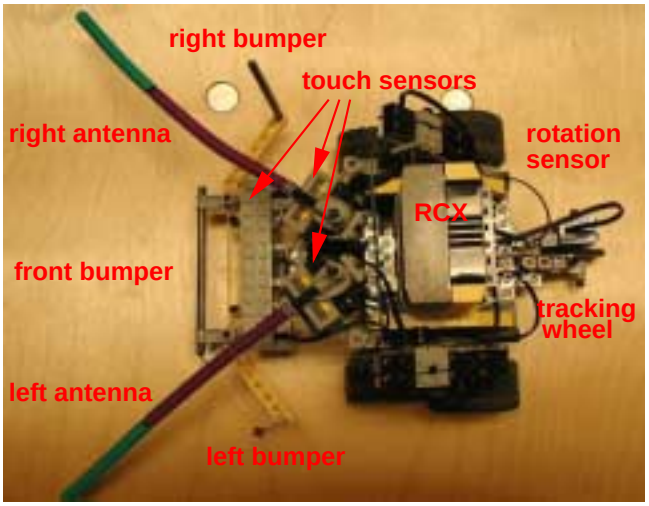

Fig. 1. Robot with antennas, bumpers, touch sensors and rotation sensor.

Figure 1 shows a top-view of the robot used in the experiments. The robot has two flexible antennas, two touch sensors per antenna detect whether the antenna bends to the left or right. In addition a left, central and right front bumper are connected to two touch sensors (mounted underneath the grey LEGO plate) that detect collisions with object. The robot is driven by four wheels, the motor direction of the left and right wheel pair can be set independently to forward, reverse and float (the motor spins freely).

The rotations of a freely spinning tracking wheel at the robot tail are counted by a rotation sensor which allows it to compute the actual the distance traveled by the robot.

The robot behavior is almost purely reactive, the RCX reads the sensor states upon which the controller executes a motor command. It is possible to specify a delay for each motor commands, which means that even if the sensor states changes in the meantime the specified motor action is not aborted prior to the delay period. By delaying left and right motor actions differently, the robot rotates by an angle proportional to the difference in delay between both actions.

5 Learning a Tactile Wall-Following Behavior

5.1 Behavior Representation

The robot controller obtains input from six binary touch sensors, which means that in principle the perception space contains $2^6 = 64$ different sensory states. However, as the two antennas can only actuate either its left or right touch sensor, the actual number of possible sensory states is 32. The actual controller only discriminated among nine different sensor perceptions $S^1, \ldots, S^9$. In state S_1, no touch sensor is active, this corresponds to a situation in which the robot has no contact with walls or obstacles. A default action of both motors running in forward direction is assumed for this state. The states S_2, S_3, S_4 capture the bumper sensors and correspond to frontal bumper pressed, left bumper pressed and right bumper pressed. The role of the antennas is to provide some remote sensing capability, they are therefore only considered in case no bumper sensor is active. The states S_5, S_6, S_7, S_8 reflect situations in which the left respectively right antenna is bend inwards or outward. Finally, the state S_9 captures the situation in which both antennas are bend outwards.

Three bits $s_1, \ldots, s_3$ determine one of the eight possible motor action associated to each state S_i. Left and right motor can independently operate in forward, reverse or floating mode. To avoid that the robot stands still, the motor actions in which both left and right motor are in floating mode is excluded. Five additional bits $s_4, \ldots, s_8$ per state specify the delays $\Delta t_l, \Delta t_r \in [0 - 70ms]$ of left and right motor action.

$$\begin{aligned} \Delta t_l &= s_4 * 40 + s_5 * 20 + s_6 * 10 \\ \Delta t_r &= s_4 * 40 + s_7 * 20 + s_8 * 10 \end{aligned} \tag{5}$$

By assigning different delays to left and right motor action the robot is able to execute controlled turns by a fixed angle.

The entire chromosome has a length of $8x8 = 64$ bits and encodes the motor actions associated to the sensor states $S_2, \ldots, S_9$. No motor action action is encoded for state S_1 as by default the robot moves forward if it makes no contact with walls or obstacles.

5.2 Fitness Function

The objective is to evolve a controller that maximizes the forward motion of the robot measured by the freely spinning wheel in the tail. Each candidate

behavior is executed over a period of 50 seconds, after which the number of rotations is returned as a fitness value. Notice, that moving backward reduces fitness as the rotation counter decrements. When the robot turns on the spot the spinning wheel slides rather than rotates and therefore the rotation counter usually neither increments nor decrements during such a motion. The theoretical maximum fitness of about 1000 rotations is obtained if the robot moves straight forward in an obstacle-free environment over the entire trial. However, as the robot possesses no remote sensing capability the theoretical maximal fitness is never achieved as collisions with walls and obstacles in the maze force the robot to turn and back-up. The robot is able to minimize evasive manœuvres by assigning appropriate motor actions to the sensor states. In addition the robot is partially able to determine which future sensory information it receives as the selected action effects its position in the environment and thereby indirectly its perception of the environment as well.

Perceptual aliasing occurs when two states that require different motor actions generate the same perception. The robot is not able to discriminate between the two situations based on the sensory information alone. Nevertheless, sensory-motor coordination allows robot to cope with perceptual aliasing by selecting those motor actions that more likely result in disambiguous sensory patterns a strategy known as active perception [2]

6 Experiments

Fig. 2. Hexagonal maze used in the experiments.

A genetic algorithm evolved the 64-bit strings that encoded the mapping from eight possible sensor states to motor actions as described in section 5.1 with a population size of ten individuals evolved over twenty generations.

The behavior parameters are down-loaded on the LEGO Mindstorm robot and the performance is evaluated in the hexagonal, looping maze of size $160x120cm$ shown in figure 2. In an earlier experiment the maze contained no obstacles which the wall-following behavior to rely solely on the bumper sensors due to the homogeneous structure of the environment. Therefore, we decided to increase the complexity of the environment by mounting cans to the walls. These additional obstacles made the wall-following task more difficult, due to an increase in perceptual aliasing .

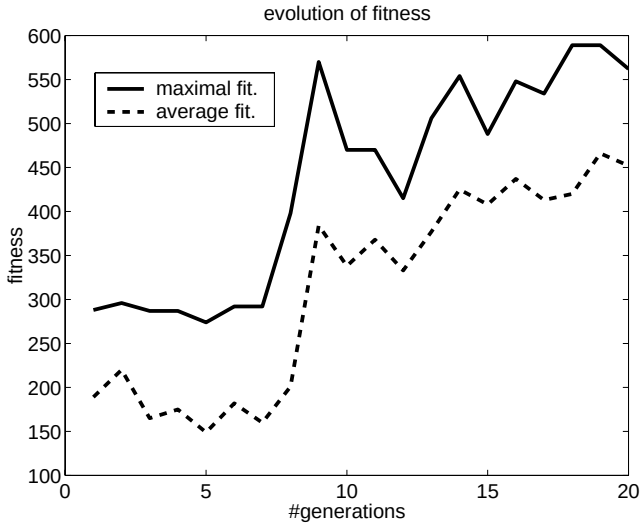

Fig. 3. Evolution of maximal and average fitness over time.

Each behavior is executed on the robot over a period of 50 seconds after which the RCX transmits the number of rotations accumulated during the trial as the fitness to the genetic algorithm running on the host computer. The genetic algorithm employed fitness-proportionate selection with linear scaling, two-point crossover with a crossover probability of $p_c = 0.7$ and mutation with a mutation rate of $p_m = 0.01$ per bit. Figure 3 shows the evolution of the average and maximal fitness with the number of generations. The best individual in the initial population achieved a a fitness of 288 rotations, the best overall behavior emerged in 18th generation and achieved 589 rotations, which corresponds to a traveled distance of about $3.5m$.

The mapping from sensor states $S_1, \ldots, S_9$ to motor actions of the best individual is shown in table 1. Notice, that in case of a left bumper collision the robot turns right (left forward, right motor in reverse) and vice versa turns right in case the right bumper is pressed. The robot reacts similar when it perceives obstacles with its tactile antennas.

A manually designed wall-following behavior achieved an average performance of about 470 rotations per trial, which is comparable with best evolved behavior considering that the performance of a behavior varies with the robot starting position in the maze.

Movies that show the robot performing an initial, typical and the optimal behavior are available at `http:\\www.nada.kth.se\~hoffmann\lego.html`

sensor state	left motor		right motor	
	direction	delay Δt_l	direction	delay Δt_r
S_1 :no contact	fwd	0 ms	fwd	0 ms
S_2 :front bumper	rev	50 ms	rev	50 ms
S_3 :left bumper	fwd	40 ms	rev	70 ms
S_4 :right bumper	rev	30 ms	fwd	30 ms
S_5 :left antenna outward	fwd	60 ms	fwd	60 ms
S_6 :left antenna inward	float	20 ms	rev	30 ms
S_7 :right antenna inward	rev	60 ms	fwd	70 ms
S_8 :right antenna outward	rev	70 ms	fwd	40 ms
S_9 :left & right antenna outward	fwd	20 ms	rev	10 ms

Table 1. Mapping from sensor states $S_1, \ldots, S_9$ to motor actions for the best behavior.

7 Conclusion and Future Work

This paper presented an evolutionary learning scheme for robotic behavior design. The genetic algorithm learned a mapping from sensor states to motor actions that to accomplish a tactile wall-following behavior. For the final paper we are going to investigate how the genetic algorithm can exploit the fitness model to reduce the number true fitness evaluations using the sampling scheme proposed in section 3.

Acknowledgments

This research presented in this paper has been sponsored by the Swedish Foundation for Strategic Research.

References

1. C.G. Atkeson, A.W. Moore, and S. Schaal. Locally weighted learning for control. *Artificial Intelligence Review*, 11:75–113, 1997.
2. R. Bajcsy. Active perception. *Proceedings of the IEEE*, 76(8):996–1005, 1988.
3. W. Banzhaf and P. Nordin. An on-line method to evolve behavior and to control a miniature robot in real time with genetic programming. *Adaptive Behaviour*, 5(2):107–40, 1997.

4. R.A. Brooks. New approaches to robotics. *Science*, 254:1227–1232, 1991.
5. D. Floreano and F. Mondada. Evolution of homing navigation in a real mobile robot. *IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics*, 26(3):396–407, 1996.
6. D.E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.
7. F. Hoffmann and G. Pfister. Evolutionary design of a fuzzy knowledge base for a mobile robot. *International Journal of Approximate Reasoning*, 17(4):447–469, 1997.
8. N. Jabobi. Half-baked, ad-hoc and noisy: Minimal simulations for evolutionary robotics. In *Proceedings of the Fourth European Conference on Artificial Life*, pages 348–357. MIT Press, 1997.
9. S. Nolfi and D. Floreano. *Evolutionary Robotics – The Biology, Intelligence, and Technology of Self-Organizing Machines*. Intelligent Robotics and Autonomous Agents. MIT Press, 2000.
10. A. Ratle. Optimal sampling strategies for learning a fitness model. In *Proceedings of the 1999 Congress on Evolutionary Computation CEC 1999*, volume 3, pages 2078–2085, 1999.

Appendix A: Presentation Plan

In addition to the paper we are going to provide the following material in our online-presentation at the web-site `http:\\www.nada.kth.se\~hoffmann\lego.html`.

- Movies and pictures of the robot during the experiment traveling the maze demonstrating the performance of the adapted wall-following behaviors.
- Illustrated step-by-step instructions how to construct the mechanical robot from LEGO parts.
- NQC source code and executable code that can be down-loaded to the LEGO robot to run the adapted behaviors.
- The source code for the genetic algorithm in C++ and the NQC code to reproduce the evolutionary learning experiments.
- Useful links to evolutionary robotics and LEGO Mindstorm robots.