On Sensor Evolution in Robotics

Karthik Balakrishnan and Vasant Honavar
Artificial Intelligence Research Group
Iowa State University
Ames, IA - 50011.
balakris@cs.iastate.edu, honavar@cs.iastate.edu
http://www.cs.iastate.edu/~{balakris,honavar}/homepage.html

Abstract

In recent years, evolutionary algorithms (EAs) have been successfully used in the design of artificial neural networks for a variety of applications. The suitability of EAs for this design task stems from their ability to adaptively search large spaces in near-optimal ways. One direct application of this advance has been in the area of evolutionary robotics, where EAs are typically used for designing behavior controllers for robots and autonomous agents. While such designs have been found to work well in general, their performance is often limited by the number, placement, quality, efficacy, and reliability of the sensors that the robots are endowed with. In this paper we argue that designing the sensory systems of these robots, in addition to the usual practice of designing the controller, can lead to improvements in the performance of the robot. Our results indicate that the evolution of sensors is a useful enterprise, and can lead to efficient and often counterintuitive controller designs.

1 Introduction

Robots and autonomous agents are equipped with control mechanisms that allow them to produce specific behaviors in any given environment. Not only must these control mechanisms be efficient and robust, but also able to adapt to a changing environment. *Artificial neural networks* are therefore viable candidates for such behavioral control mechanisms [7, 5, 2].

However, realizing even the simplest of robotic behaviors requires fairly complex trade-offs among several, often conflicting, objectives. For example, to implement *wall-following* behavior, a robot has to optimize multiple criteria like efficient sensing, recognition of the wall, reasonably precise estimation of the distance to the wall, maintenance of a safe distance to avoid bumping into the wall (while keeping it within sensor range), appropriate control of speed, etc. Thus, design of appropriate neural network controllers (or neurocontrollers) to realize such behaviors is an instance of a difficult multi-criterion optimization problem.

Evolutionary Algorithms (EAs), simulated models of natural evolution, have been shown to be effective procedures for searching large, complex, multi-modal, and deceptive spaces [8, 6], and can thus be employed to solve multi-criterion optimization problems like the design of neurocontrollers for robots and autonomous agents [9, 12, 5, 10, 3]

Although a large body of work exists in this area, most of them assume a fixed sensory system architecture, while concentrating only on the design and development of the control mechanism. For example, the miniature robot Khepera [11] was used in the navigation studies in [5]. Since Khepera comes equipped with eight infra-red (IR) sensors, the neurocontrollers evolved in [5], assumed the availability of eight IR sensors. Fewer (or more) sensors might have resulted in better performance, but that dimension was not explored.

The wide variety of sensory systems represented in nature are suggestive of the power of evolution in determining good designs over many such dimensions, including — sensors of different *types* (e.g., vision, tactile, olfactory etc.), *characteristics* (e.g., range, sampling frequency, response function etc.), *numbers* (e.g., two eyes, one nose etc.), *placement* (e.g., front, rear, top etc.), modes of *sensory information integration* (e.g., through the use of appropriate neural circuits), etc.

Just as specific designs of the sensory system provide the endowed biological organisms with a survival advantage, it is natural to wonder if the performance of robots and autonomous agents would benefit from similar sensory system design considerations. This paper attempts to answer this question by using evolutionary techniques to discover good designs of sensory systems.

2 The Simulation Task

The robotic simulation that we consider in this paper is based on a task proposed by Teller [13]. Here the robot is placed in a square arena of $N \times N$ cells, in which M boxes are randomly scattered in the inner $(N-2)\times(N-2)$ grid. The arena is enclosed by *impenetrable* walls as shown in Figure 1.

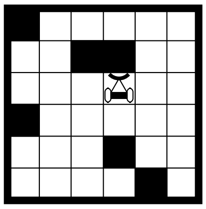

Figure 1: **The operating environment. Shaded squares represent boxes and unshaded ones, empty space.**

The robot is started off in a random position facing one of four directions — north, south, east, or west; with the task of clearing the arena by moving the boxes to the sides. As per the original specification of Teller, the robot has eight sensors which provide the robot with specific inputs so that it can detect the presence of boxes, walls, or empty spaces. The sensors are *fixed* to the robot (i.e., turn with the robot), and *sense* one square in each of the eight directions surrounding the robot. The robot determines its output action based on the information provided by the sensors, and possibly the history of its past actions.

The robot is capable of three possible actions — forward move, 90 degree anti-clockwise turn, and 90 degree clockwise turn. Turns are in-place, while move forward actions result in the robot moving ahead by one square. If there happens to be a box immediately in front of the robot when it moves forward, the box gets pushed ahead by one square. This is the primitive operation that the robot needs to clear the arena.

Attempts to push a box fail under two circumstances: the box may already be against a wall; or there may be another box adjacent to it in the direction in which the box is being pushed. The latter is tantamount to saying that the robot has the capacity to push *at most one* box at a time. The fitness of the robot is measured by the number of boxes it pushes to the walls within an allocated period of time. Our objective then, is to design a neurocontroller for this robotic bulldozer that allows it to attain a high fitness. We would also like to determine if the fitness might be improved upon by a judicious design of the number, placement, and range of sensors. For reasons mentioned in section 1, we use EAs to search for such designs.

3 Implementation Details

In our simulations we use *Genetic Algorithms* (GAs) [8, 6] to evolve sensory and neurocontroller designs. Our *genetic representation* had two parts, one encoding the placement and ranges of the sensors, and the other encoding the input connectivities of the units of the neurocontroller, as shown in Figure 2. The sensors were encoded using a 2-tuple (i, j), with the understanding that the corresponding sensor is placed to detect the square i cells *behind* and j cells to the *right* of the cell occupied by the robot. Thus, negative values of i mean the sensor is *frontal*, i.e., in front of the robot, while negative $j's$ indicate that the sensor is positioned to the *left* of the robot. For example, a value of (-1,0) encodes a sensor that senses the square one cell to the *north* and zero cells to the *east* of the current position of a *north-facing* robot (labeled **N** in Figure 2). This sensor would detect the cell to the east if the robot turns clockwise through 90 degrees. (Henceforth, sensors will be labeled assuming a north-facing robot, with the understanding that the sensors turn with the robot.)

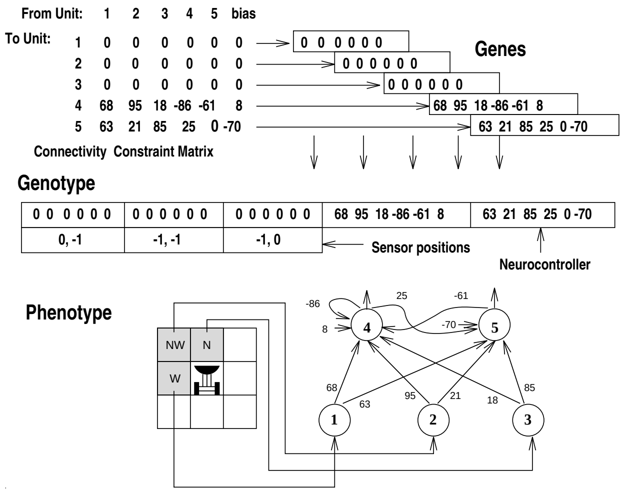

Figure 2: **Genetic representation used.**

The genetic operator *crossover* was defined to swap *logical blocks* (genes), where each block is the specification of either one sensor or one unit. *Mutation* was defined to randomly change values in the blocks, thereby producing changes in either the placement/range of sensors or in the input connections of a unit (depending on what the corresponding block encoded). Our simulations used populations of size 100, and the evolutionary runs lasted 100 generations. We used *uniform-crossover* with crossover probability of 0.5, and mutation with probability 0.1 per gene. The weights in the network were constrained to be *integers* in the range [-100$\cdots$+100], while sensor positions and ranges were bounded by the size of the arena.

Each sensor provides one of three input values to the corresponding input unit of the neurocontroller: 1 for a *box*, 0 for *empty space*, and -1 when it detects a *wall*. At each time-step, the network updates its inputs to reflect the sensor readings and each neuron in the network computes its output (by applying the threshold function to its summed input) and passes it on to each of the neurons to which it is connected. Thus, at each time step, signals are propagated through a depth of one link through the network. The output of the network at each time-step results in an (instantaneous) action by the robot. The neurocontrollers have two output units, one codes for the *action* (move vs turn), while the other determines the *direction of turn*. When the action unit produces a +1, the robot moves forward, while a -1 forces it to turn. On a turn action, the actual direction of turn is given by the output of the turn unit (-1 for clockwise and +1 for anti-clockwise).

The general framework for the simulation experiments was borrowed from [13]. Thus we used 6×6 arenas, with 6 boxes randomly distributed in the inner 4×4 grid. In each trial, the robot was placed in a randomly chosen empty cell, facing a random direction and allowed to perform $O(N^3) = 216$ actions (where each action takes one time step). For reasons mentioned in [3], Teller's simulation time estimate of $O(N^2) = 80$ was not used.

At the end of 216 time steps each box against the wall was awarded one point, while boxes in the corners fetched one extra point. This was deemed the fitness of the robot in the current trial. The overall fitness of the robot was evaluated by averaging its fitness over 100 random trials, where each trial involved placing it in a random environment and simulating it through 216 time steps as described above. This fitness value was used by evolution in the *fitness-proportionate selection* of parents.

4 Simulation Results

We report simulation results for the evolution of neural networks of the *initially feedforward* class. By this we mean that evolution starts off with feedforward networks, but through low probability mutation, retains the ability to add recurrent links, if need be. Our earlier work shows the empirical usefulness of such an approach in designing neural networks with near-minimal recurrence [3]. The absence of a link is represented by a zero value at the corresponding weight position (see, for example, Figure 2). For the purposes of keeping the analysis simple and the discussion focused, we have chosen to explore networks without any hidden units, although the addition of hidden units is seen to improve performance considerably [1]. In addition, only one kind of sensor is simulated in our experiments and the information acquisition by the sensors is assumed to be fault-free. Though this is an oversimplification, it can be addressed more rigorously by building realistic models of sensor noise and failures.

4.1 Experiment 1: Fixed Sensor Positions

In this set of experiments, evolution explores only the neurocontroller design space (while keeping the sensory system fixed). Thus, the robot had eight sensors placed to sense the eight immediate squares surrounding the robot, as shown in Figure 3.

We performed five evolutionary runs, each starting off with a different random seed (hence a different initial random population). The average fitness of the best network evolved in the five runs was 3.74, while the best network ever found had a *labeled fitness* of 5.03.

Since the fitness evaluations during evolution are based on averaging the fitness over 100 random environments, the fitness figures reported by different evolutionary runs are not truly comparable (as the 100 random environments might have been different in each run). Hence we took the best network found in each run, and reevaluated it over a chosen set of 500 environments. This fitness figure, which we call *corrected fitness*, is 4.07 for the best network. This also happens to be the network with the best labeled fitness (5.03), and is shown in Figure 3.

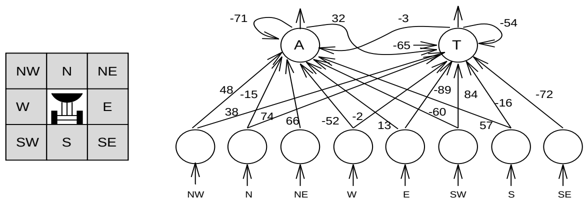

Figure 3: **Evolution of neurocontroller alone. The figure on the left shows the eight active sensors. Also notice the large negative self-loop at the action unit A.**

As observed in [3], the evolved neurocontrollers have a large negative self-loop at the output unit coding for action (labeled **A** in the figure). Such an arrangement biases the robots to interleave actions of moving forward and turning, thereby preventing them from getting into states in which they might be stuck till the end of simulation. This feature allows such robots to achieve higher fitnesses on this simulation task [3].

4.2 Experiment 2: Evolving Sensor Positions

In this experiment, evolution was allowed to operate not only on the structure of the neurocontroller, but also on the placement of the sensors. In order to determine if Teller's configuration of eight sensors [13], could be improved upon, we allowed evolution to start with random initial networks with the eight sensors placed as in Experiment 1. The sensor positions were incorporated as eight additional genes in the genotypic representation used in the above experiment. Thus, mutation could *move* sensors and crossover could exchange sensor positions between parents. We also incorporated a primitive *sensor switch* — where with *extremely low probability*, mutation could switch off sensors, thereby nullifying their effect on the behavior of the robot. The intuition here was that the evolutionary system might be forced to *co-optimize* the effective number of sensors in addition to their placement, although *no penalty was attached to the number of sensors used.*

The average fitness of the best network produced in five random runs was found to be 3.98. Even though there is little improvement in fitness over Experiment 1 (3.74), it is interesting to note that in each of the runs *at least one* sensor was switched off. In particular, one of the runs turned off *five sensors* yet attained among the highest fitnesses observed. The corresponding network and sensor placement are shown in Figure 4. This network, with a labeled fitness of 4.95, results in a corrected fitness of 4.26 (when evaluated on the *same* set of 500 random environments used in Experiment 1). One noticeable feature of this robot is that its active sensors are on its left. Thus, the robot should be more prone to clockwise turns, which would keep the object within sensor range even after a turn. This is visually confirmed by the robot runs.

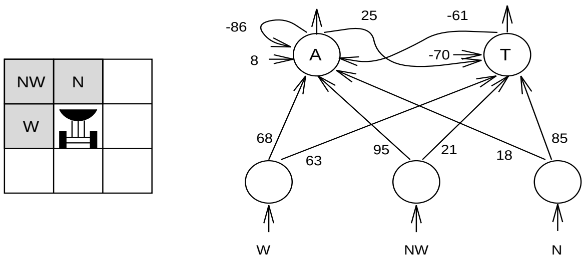

Figure 4: **Evolving controller and sensor positions. The most compact network found — only three sensors are used, the other five are discarded by evolution.**

The best network produced in this experiment had a labeled fitness of 5.41 and corrected fitness 4.74. This network, shown in Figure 5, uses only six of the eight sensors available. Further, evolution designs the network with two sensors *each* placed at the North-East (NE) and East (E) sides of the robot, thereby allowing the combined weights to exceed the [-100· · ·+100] *limit imposed by our simulation design* (in the case of the NE sensor).

The networks evolved in this experiment were observed to have the same basic design as those in Experiment

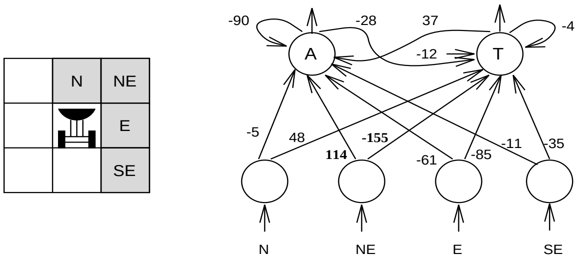

Figure 5: **Evolving controller and sensor positions. The best network discovered by evolution. Through the placement of two sensors at the NE position, evolution discovers a way of exceeding the [-100· · ·+100] limit on the weights.**

1, including the large negative self-loop at the output unit coding for action. Further, even though there was no pressure to optimize the number of sensors, evolution produced designs using fewer sensors than the eight available. It is interesting to consider possibilities of penalizing the number of sensors so as to bias evolutionary search in favor of designs with fewer sensors.

4.3 Experiment 3: Evolving Sensor Ranges

What would happen if evolution were allowed to tune the ranges of the sensors also? Would this lead to more effective designs? This experiment attempts to explore these questions.

Since all physical sensors have a maximum limit on their range, for the purposes of this study we fixed the maximum range of the simulated sensors to two cells. Note that we measure distances in terms of the number of intervening cells, not any physical metric like the Euclidean distance etc. For example, the sensor labeled 2NW in Figure 6 is considered to be two cells away from the position occupied by the robot.

In this experiment, evolution could not only rearrange the sensor placements, it could also tune their sensory ranges to zero, one, or two cells. A zero range is equivalent to a switched off sensor. Based on the evolutionary runs, the average fitness of the best networks evolved was found to be 4.28, while the best network produced by any one evolutionary run had a fitness of 5.35 (corrected fitness 4.66). The corresponding network architecture and sensor placement are shown in Figure 6.

Even in this case, the robot interleaves the move and turn motions (suggested by the weight of -100 at the action unit). In addition, this robot has two of its sensors placed behind it — a design that is highly counterintuitive considering the fact that the robot cannot move backwards. Observation of its actual behavior suggests that the robot uses its NE and SE sensors to move/orient itself such that the box is detected by the E sensor. At

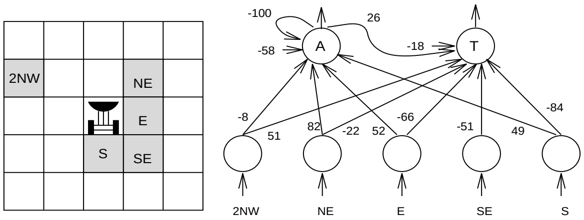

Figure 6: **Evolving controller, sensor positions, and range. The 2NW sensor allows the robot to *explore*.**

this point the robot turns towards the box, and even though it does not have a N sensor, manages to push the box given its tendency for interleaving turns and moves. The function of the 2NW sensor is twofold. When it detects the presence of a wall it suggests a *clockwise* turn to the neurocontroller, thereby moving the robot away from the wall. If it encounters a box, however, it helps the robot to turn in an *anti-clockwise* direction, i.e., towards the box. It is interesting to notice that this design too, uses fewer than the eight sensors initially provided.

5 Discussion

The results presented in this paper argue in favor of evolutionary design techniques that simultaneously explore the space of designs for sensory systems as well as controllers.

In our simulations, we observed that robots of higher fitness are produced if sensor placement and their ranges are also designed for the specific task. Further, the evolutionary system produced controllers utilizing far fewer sensors than the number actually made available. This is particularly interesting because these designs developed in the absence of any kind of penalty on the number of sensors used. It remains to be seen if explicit penalties on the number of sensors would lead to even more efficient designs.

The main limitation of the designs that were discovered in these experiments is the relatively low fitness of the robots — even the best robots are unable to clear the arena completely. The primary reason for this shortcoming is that the robots often get stuck in repetitive/cyclic paths. This can be handled by introducing noise in the system [4], thereby allowing the robots to break out of such fixed cycles. As some researchers have argued [4, 10], such noisy systems are of particular importance if controller designs evolved in simulation are to be successfully transferred on to real robots.

Sensory system design, in our view, has two main applications. First, it allows us to determine efficient, often counterintuitive designs for the control mechanism, as was seen in Experiments 2 and 3. Further, even though this paper assumed all sensors to be of the same kind, the EA-based design strategy could be easily extended to cope with sensors of different types. It could thus serve as a potentially useful approach to alleviate the problem of *sensor fusion* (how to combine/integrate information from multiple sensors, possibly of different kinds) in robotics.

Second, sensory system design is a necessary component in the development of robust, fault-tolerant systems. For example, many robots and autonomous agents are designed to operate in hazardous and unpredictable environments (nuclear waste cleanup, toxic chemical handling processes, space exploration etc.). In such applications, sensor failures might occur quite frequently. Hence there is a strong need to design systems that can operate reliably in such scenarios. This may be achieved by modeling the environment, including sensor failures, and performing studies as in Experiment 2 to evolve fault-tolerant sensor placements and appropriate controller designs.

These and other issues (e.g., fitness evaluation by placing the *real* robots directly in the physical environment of interest) are topics of ongoing research.

Acknowledgments

Vasant Honavar would like to thank the National Science Foundation for partial support of this research through grant NSF IRI-9409580.

References

[1] K. Balakrishnan and V. Honavar. Analysis of neurocontrollers designed by simulated evolution. Technical Report CS TR 95-25, Department of Computer Science, Iowa State University, Ames, IA - 50011, November 1995.

[2] K. Balakrishnan and V. Honavar. Evolutionary design of neural architectures — a preliminary taxonomy and guide to literature. Technical Report CS TR 95-01, Department of Computer Science, Iowa State University, Ames, IA - 50011, January 1995.

[3] K. Balakrishnan and V. Honavar. Analysis of neurocontrollers designed by simulated evolution. In *Proceedings of IEEE International Conference on Neural Networks ICNN'96*, 1996. To appear.

[4] D. Cliff, P. Husbands, and I. Harvey. Analysis of evolved sensory-motor controllers. In *Second European Conference on Artificial Life*, 1993.

[5] D. Floreano and F. Mondada. Automatic creation of an autonomous agent: Genetic evolution of a neural-network driven robot. In D. Cliff, P. Husbands, J-A Meyer, and S. Wilson, editors, *From Animals to Animats 3: Proceedings of the Third Inter-*

national Conference on Simulation of Adaptive Behavior, pages 421–430, Cambridge, MA, 1994. MIT Press.

[6] D. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, Reading, MA, 1989.

[7] I. Harvey, P. Husbands, and D. Cliff. Issues in evolutionary robotics. In J. Meyer, H. Roitblat, and S. Wilson, editors, *From Animals to Animats II: Proceedings of the Second International Conference on Simulation of Adaptive Behavior*, pages 364–373, Cambridge, MA, 1992. MIT Press-Bradford Books.

[8] J. Holland. *Adaptation in Natural and Artificial Systems.* The University of Michigan Press, Ann Arbor, 1975.

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

[10] O. Miglino, K. Nafasi, and C. Taylor. Selection for wandering behavior in a small robot. *Artificial Life*, 2(1):101–116, 1994.

[11] F. Mondada, E. Franzi, and P. Ienne. Mobile robot miniaturization: A tool for investigation in control algorithms. In *Proceedings of the Third International Symposium on Experimental Robotics*, Kyoto, Japan, 1993.

[12] S. Nolfi, J. Elman, and D. Parisi. Learning and evolution in neural networks. *Adaptive Behavior*, 3(1):5–28, 1994.

[13] A. Teller. The evolution of mental models. In Kim Kinnear, editor, *Advances in Genetic Programming*, pages 199–219. MIT Press, Cambridge, MA, 1994.