Crossroads
The ACM Student Magazine

Evolutionary learning in mobile robot navigation

by ***Cory Quammen***

Introduction

As humans, we enjoy the luxury of having an amazing computer, the brain, and thousands of sensors to help navigate and interact with the real world. The product of aeons of evolution has enabled our minds to model the world around us based on the information gathered by our senses. In order to navigate successfully, we can make high-level navigation decisions, such as how to get from point A to point B, as well as low-level navigation decisions, such as how to pass through a doorway. The brain's capacity to adapt has also made it possible for people without certain sensory capabilities to navigate throughout their environments. For example, blind people can maneuver through unfamiliar areas with the aid of seeing-eye dogs or canes. Even without all of our sensors, we are able to cope with familiar and unfamiliar environments.

The abilities of modern robots stand in stark contrast to human abilities. Robots have far fewer sensors and a far less sophisticated computer; consequently, they have great difficulty adapting to changes in their capabilities and environments. Common types of robotic sensors include touch sensors for collision detection and sonar sensors for detecting obstacles at a distance. Some highly sophisticated robots use stereo vision to determine information about their environment, but this requires solving the substantial problem of computer vision first. Alas, most robots "experience" their environments through rudimentary sensors, and must process the limited information these sensors provide in order to make low-level navigation decisions. The problems of robot navigation are intensified if certain sensor capabilities are lost due to malfunction or damage incurred during operation.

To better understand the limitations of a robot, imagine a situation where you are in an unknown house with only vague directions from your current location to the kitchen. Next, you put on a hard plastic suit that completely isolates you from the outside world. Inside the suit, all you can hear is a beeping sound that beeps faster as you approach walls or other obstacles in your path. You must use the beeping and vague directions to successfully navigate to the kitchen. Chances are, after some stumbling around and banging into walls, you will be able to find your way there.

Likewise, a successful robot navigation system will "find its way to the kitchen" using a full complement of sensors. Ideally, the system would also adapt to sensor malfunctions. In fact, researchers have developed several approaches to handling sensor failure including the use of Bayesian networks [**2**], case-based reasoning [**3**], and evolutionary learning [**10**]. This article describes how evolutionary learning using simulations of a robot and its environment can be effective in solving real-world robot navigation problems, including (sensor) failure handling.

Evolutionary algorithms

Charles Darwin first identified the process of natural selection in his monumental work *The Origin of Species*. Certain characteristics that govern an individual's a chance of survival are passed to offspring during reproduction. Individuals with poor characteristics die off, making the species stronger in general. Inspired by this natural process of "survival of the fittest," **evolutionary algorithms (EAs)** attempt to find a solution to a problem using simulated evolution in a computer.

There are two related, yet distinct, types of EAs of particular interest to this article. The first type, **genetic algorithms (GAs)**, involve manipulating a fixed-length bit string. The bit string represents a solution to the problem being solved; it is up to the programmer to determine the meaning of the string. The second, **genetic programming**, involves generating expression trees as used by languages such as Lisp and Scheme. With genetic programming, actual programs can be created and then executed.

Generally speaking, the process used in evolutionary learning begins by randomly generating a population of individuals where each **individual** is a potential solution to the problem. The **population** is the set of individuals generated. Each individual contains a genome, the content produced during the execution of EAs. In the case of GAs, the fixed-length bit string is the genome.

Next, each individual in the population is evaluated using a fitness function. **Fitness functions** use a method of testing solution quality for the respective problem domain. For example, if a genetic algorithm were designed to determine a single-variable algebraic function F(n), the fitness function may use a training set of inputs and outputs to determine how close an individual is to F(n). In this case, the fitness could be inversely proportional to the sum of the absolute differences between expected output and actual output for each input.

How the fitness of an individual is used depends on the implementation of the evolutionary algorithm. Usually, a higher fitness value corresponds to a greater chance of the individual being selected for reproduction. Different methods of producing the next generation exist, but most commonly, two operators are employed: **mutation** and **breeding**. Mutation involves randomly altering one or more genes in an individual's genome. Breeding uses a **crossover** operation to combine components of two parents' genomes to produce one or more children. Once a new generation is created, the old one is discarded [**1**].

A cycle of evaluation and reproduction is repeated through several generations, as specified by the programmer. The evolution process ceases after a termination criterion has been met, and the result of the evolution run is the individual found to be best-so-far [**5**]. For a thorough introduction to GP, consult [**1**].

A summary of the components needed for EAs follows as in [**7**]:

- A genetic representation for potential solutions to the problem (i.e. fixed (or variable)-length bit string, expression tree)
- A method to create an initial population of individuals
- A fitness function that plays the role of the environment, rating solutions in terms of their "fitness"
- Operators that determine the composition of children
- Values for various parameters, which the evolutionary algorithm uses (solution set size, probabilities of applying genetic operators, etc.)

Applying EAs to Robotic Navigation: Continuous and Embedded Learning

Evolutionary algorithms have been implemented to solve problems in robot navigation.

In particular, EAs have been used to get a robot to learn how to adapt to its limited capabilities. Using GP in this way is termed **evolutionary learning**. As stated before, robot navigation is a difficult problem to solve, and it becomes increasingly more difficult when a robot encounters a failure in either its **sensors**, devices that tell a robot about its environment, or its **actuators**, devices that allow a robot to physically interact with its environment.

There are numerous examples of EAs being applied to robotic mobility. The GOLEM project has evolved robots that produce novel means of locomotion in simulation, and which have later been realized using computer-aided manufacturing [**6**]. Miglino, Lund, and Nolfi have successfully evolved navigation behaviors using highly accurate simulations for training and applied the results to a Khepera miniature mobile robot [**8**]. Wu, Schultz, and Agah have used GP to develop robot behaviors for distributed micro air vehicles in simulation [**11**].

Schultz and Grefenstette have applied evolutionary learning to robot navigation with an approach they call Continuous and Embedded Learning (CEL) [**10**]. CEL's reliance on simulation of the robot and its environment to evolve navigation behaviors, is similar to other applications of evolutionary learning. Schultz and Grefenstette's work extends that of Miglino, Lund, and Nolfi by enabling a robot to dynamically change its simulation of its own sensory capabilities rather than having a static simulation of sensory capabilities explicitly detailed by the developer [**8**, **10**].

CEL is different from other EA applications in some respects. In most other EA applications, two distinct steps occur: an initial training period is conducted by running the EA on a training set, followed by the execution of the best-fit solution. With CEL, the two steps are linked and operate concurrently while the robot is performing its task. Thus, the learning process is continuous. The system has two major components, the learning system and the execution system. The learning system constantly evolves new behaviors in the test knowledge base and evaluates them against its simulation model. If the learning system determines that a new behavior would be more effective than the current one, the active knowledge base is updated in the execution system. In the execution system, the decision maker uses the active knowledge base to make navigation decisions. A monitor is included in the execution system to update the simulation model used by the learning system. If a major change in robot capability is detected by the monitor, the learning system will restart and begin learning new behaviors that will be effective in terms of the robot's capabilities. Figure 1 shows an

outline of this approach [10].

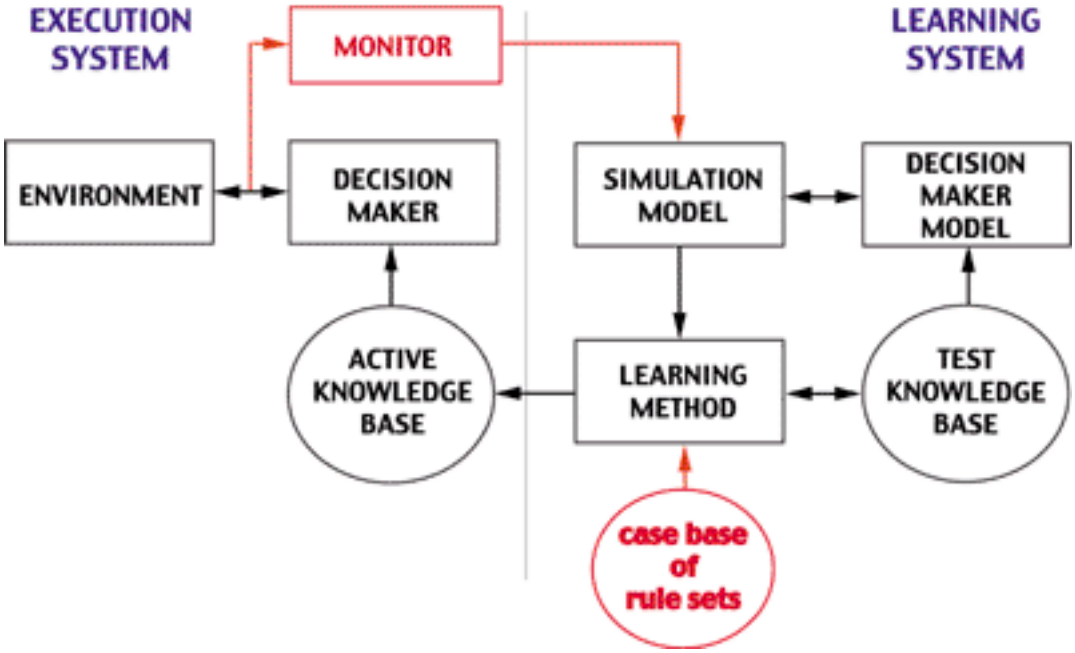

Figure 1. The Continuous and Embedded Learning Model [10].

Key components of the model are:

- Learning continues indefinitely, allowing adaptation to sensory failure.
- Learning is done on a simulation model.
- The simulation model is updated to reflect changes in the real robot or environment.

At the beginning of operation, the robot uses a case base of rule sets defined by the programmer. Each case contains an action to execute when the sensors of the robot match the parameters specified in that case. A typical rule might be:

`IF range = [10,20] AND` *`front_sonar`* `< 50 AND` *`left_sonar`* `> 20 THEN SET turn = 5 (Strength 0.7)`

Essentially, the system is based on stimulus-response where the robot reacts to the given sensor input in real-time based on a set of rules. When conditions are matched in the IF statement, the actuators will execute movement based on the parameters set in the THEN clause. One can think of these rules as genes in an EA implementation. The individual in this case is a rule set, and a base of rule sets is the population. Fitness is measured by the completion of the goal in a certain amount of time. The top 50% of individuals reproduce, producing two offspring each [10]. The learning system uses

SAMUEL, a program for evolving decision-making rules using EAs and other competition-based heuristics [4].

Performance

In Schultz and Grefenstette's experiment, a robot was given the task of navigating to the opposite side of a room through a passage in a wall starting from one wall and heading in a random direction from -90 degrees to 90 degrees, with 0 pointing directly to the opposite wall. An example course is shown in Figure 2. The measure of success of CEL was the percentage of times the robot successfully reached the goal without bumping into walls along the way or running out of time.

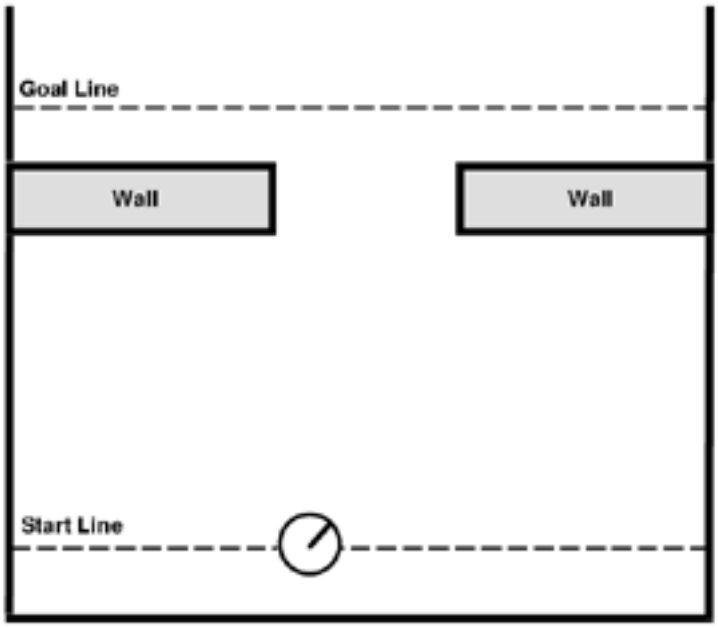

Figure 2. An example course plan used in this experiment [10].

A Nomadic Technologies Nomad 200 Mobile Robot was used for the real-world tests. The robot is a three-wheeled synchronized-steering unit that features seven front-mounted sonar units. The monitor in the execution system combines the output of sensors with information about the robot's actuator execution to determine sensor failures. If a frontal sonar unit continuously outputs zero as the distance from the robot to an obstacle but the robot keeps moving forward, then that sensor is marked as having failed. In the experiment, sonar "failure" was simulated by covering the sensor with a hard material [10].

The initial rule-set used in the experiment gave the robot a basic notion of how to get

to the other side of the room, but did not take into account obstacles or walls. The simulation model initially assumed all sensors were working [**10**].

With no evolution and all sensors functioning, the robot was able to navigate successfully about 25% of the time. After 50 generations of evolution, the robot's success rate increased to about 61%. With three sensors on the right side of the robot disabled after 50 generations, the success rate dropped to about 42%, but increased over 50 more generations to about 63%. Figure 3 shows the learning curve. These percentages are averages [**10**].

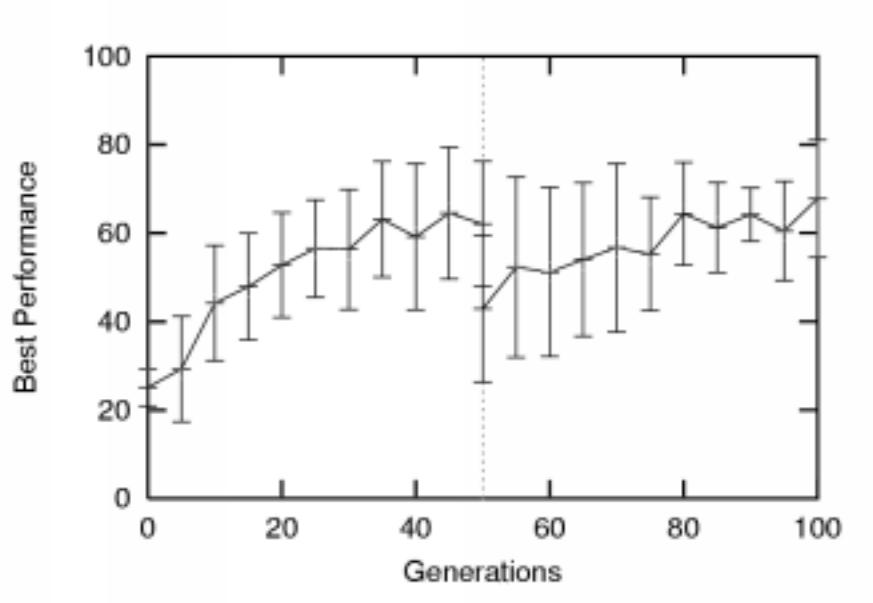

Figure 3. This learning curve shows the robot's successful adaptation to sensor failures after a certain number of generations were evolved in the learning system. The dashed vertical line at generation 50 indicates the instance when three of the front sensors were disabled. Results are averages over 10 runs [**10**].

The results of the experiment show that a robot using CEL can not only learn how to improve its navigation abilities by itself, but also re-learn how to navigate after suffering the loss of some sensory capability. While the results may not be terribly impressive, improvement in navigation and adaptation to changes in capability are clearly shown.

Conclusions

Schultz and Grefenstette's experiment showed that CEL is a viable approach to producing adaptive behavior in autonomous robots in certain domains. Robots that suffer some unpredictable failure in the field can adapt to their new capabilities using

CEL and continue to perform their tasks. Future investigations of CEL include combining sensor and actuator failures to determine how successful this approach can be and determining the limits of adaptability of CEL [**10**]. Other research is needed to improve the dynamic modeling of a robot and its environment.

In certain domains, CEL may not be an effective technique. For example, in city driving, CEL may be too slow to evolve a behavior for avoiding pedestrians. A system that combines high-level reasoning with the low-level stimulus-response behaviors that CEL is capable of developing may be more effective for a broader range of domains. However, CEL is a promising initial step in developing adaptive mobile robots that can learn on their own.

Acknowledgments

I would like to thank Alan C. Schultz for permission to discuss his work in *Crossroads*, to use the original figures in his work, and for reviewing this article. Thanks also to Professor David Wolfe and to the anonymous reviewers for helpful suggestions in writing this article.

References

1

Banzhaf, W., Nordin, P., Keller, R., Francone, F. *Genetic Programming: An Introduction*. Morgan Kauffman Publishers. San Francisco, 1998.

2

Forbes, J., Huang, T., Kanazawa, K., Russell, S. The BATmobile: Towards a Bayesian Automated Taxi. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1995.

3

Fox, S., Leake, D. Modeling Case-based Planning for Repairing Reasoning Failures. In *Proceedings of the 1995 AAAI Spring Symposium on Representing Mental States and Mechanisms*. AAAI, 1995.

4

Grefenstette, J., Ramsey, C., Schultz, A. Learning Sequential Decision Rules Using Simulation Models and Competition. In *Machine Learning* **5**(4), pp. 355-381, 1990.

5

Koza, J. *Genetic Programming II: Automatic Discovery of Reusable Programs*. The MIT Press. Cambridge, Massachusetts, 1994.

6

Lipson, H., Pollack, J. *The GOLEM (Genetically Organized Lifelike Electro Mechanics) Project.* <**http://www.demo.cs.brandeis.edu/golem/**> (17 April 2001).

7

Michalewicz, Z. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, New York, 1999.

8

Miglino, O., Lund, H., Nolfi, S. Evolving Mobile Robots in Simulated and Real Environments. In *Artificial Life*, The MIT Press, Cambridge, Massachusetts, 1995, Vol. 2, No. 4, pp.417-434.

9

Nomadic Technologies. (3 October 2000). < **www.robots.com**>(17 April 2001).

10

Schultz, A., Grefenstette, J. Continuous and Embedded Learning in Autonomous Vehicles: Adapting to Sensor Failures. In *Proceedings of SPIE* Vol.4024, pg. 55-62, 2000.

11

Wu, A., Schultz, A., Agah, A. Evolving Control for Distributed Micro Air Vehicles. In *Proceedings of the IEEE 1999 Conference on Computational Intelligence in Robotics and Automation* (CIRA-99), November 8-9, 1999.