Learning Robot Behaviors Using Genetic Algorithms

Alan C. Schultz
United States Naval Research Laboratory

LEARNING ROBOT BEHAVIORS USING GENETIC ALGORITHMS[†]

ALAN C. SCHULTZ

Navy Center for Applied Research in Artificial Intelligence
Naval Research Laboratory
Washington, DC 20375-5337
schultz@aic.nrl.navy.mil

ABSTRACT

Genetic Algorithms are used to learn navigation and collision avoidance behaviors for robots. The learning is performed under simulation, and the resulting behaviors are then used to control the actual robot.

THE LEARNING PARADIGM

The approach to learning behaviors for robots described here reflects a particular methodology for learning via a simulation model. The motivation is that making mistakes on real systems may be costly or dangerous. In addition, time constraints might limit the number of experiences during learning in the real world, while in many cases, the simulation model can be made to run faster than real time. Since learning may require experimenting with behaviors that might occasionally produce unacceptable results if applied to the real world, or might require too much time in the real environment, we assume that hypothetical behaviors will be evaluated in a simulation model (the off-line system). As illustrated in Figure 1, the current best behavior can be placed in the real, on-line system, while learning continues in the off-line system [1].

The learning algorithm was designed to learn useful behaviors from simulations of limited fidelity. The expectation is that behaviors learned in these simulations will be useful in real-world environments. Previous studies have illustrated that knowledge learned under simulation is robust and might be applicable to the real world if the simulation is more *general* (i.e. has more noise, more varied conditions, etc.) than the real world environment [2]. Where this is not possible, it is important to identify the differences between the simulation and the world and note the effect upon the learning process. The research reported here continues to examine this hypothesis.

The next section very briefly explains the learning algorithm (and gives pointers to where more extensive documentation can be found). After that, the actual robot is described. Then we describe the simulation of the robot. The task

† Appeared in Procs. of the International Symposium on Robotics and Manufacturing, August 14-18, 1994.

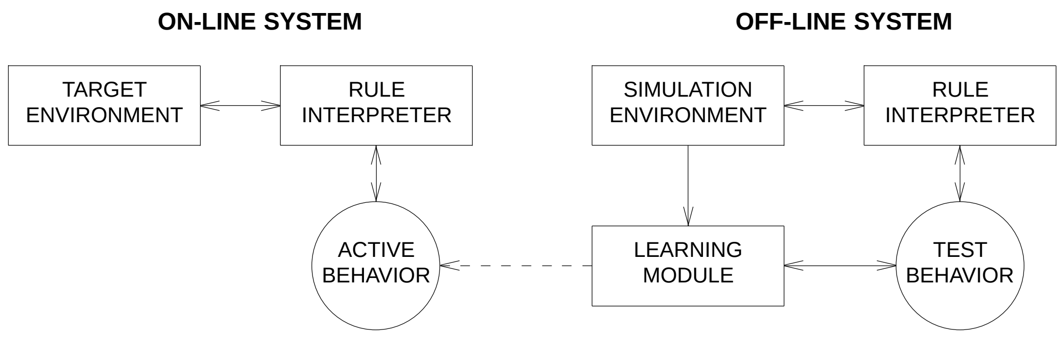

Fig. 1. A Model for Learning from a Simulation Model.

the robot is to perform is explained, followed by a description of the experiments and the results.

DESCRIPTION OF THE LEARNING ALGORITHM

Genetic algorithms, the heart of the SAMUEL system used in these experiments, are adaptive search techniques that can learn high performance knowledge structures. The genetic algorithm's strength comes from the implicitly parallel search of the solution space that it performs via a population of candidate solutions. In SAMUEL, the population is composed of candidate behaviors for solving the task. SAMUEL evaluates the candidate behaviors by testing them in a simulated environment, here a simulation of the robot and its environment. Based on the behaviors' overall performance in this environment, genetic and other operators are applied to improve the performance of the population of behaviors. One cycle of testing all of the competing behaviors is referred to as a *generation*, and is repeated until a good behavior is evolved.

The representation of a behavior is a set of stimulus-response rules. The left hand sides of the rules are matched against the sensor state of the robot, and the right hand sides of the rules specify the action the robot is to take. Behaviors are evaluated in a production system interpreter that interacts with the robot (either real or simulated). Here is an example of part of a rule set:

```
IF   front_sonar < 30   AND   bearing > 10   THEN turn = 20
IF   front_ir < 5   THEN   speed = -10
IF...
```

Each decision cycle, the interpretor reads the current state of the sensors to find rules that match, resolves conflicts to determine one rule to fire based on previously observed utilities of rules, and then fires the rule by passing back to the robot the action to be performed (for example a velocity mode command). This cycle is repeated until the task is accomplished or failed. A behavior is tested multiple times and a measure of performance is determined for that behavior. This performance measure is based on differential payoff for different behaviors of the robot. In these experiments, getting to the goal quickly yields the highest payoff while behaviors that result in collisions with an obstacle receive the lowest payoff.

One benefit of the representation language is that it allows the learning system to be easily seeded with initial knowledge [3], allowing that knowledge to be further refined. Furthermore, because of the nature of the genetic algorithm, the initial knowledge does not have to be very good; it only needs to make the system have an occasional success at performing the task. Current research is examining different ways to use existing knowledge [4].

The SAMUEL system has been used to learn behaviors for controlling simulated autonomous underwater vehicles [5], missile evasion [6], and other simulated tasks. This paper reports the first tests of the learned knowledge on a real physical system. For more details of the SAMUEL system, see [7].

ROBOT PLATFORM

For these experiments, a Nomadic Technologies, Inc. Nomad 200 robot was used. The robot's drive system, which is housed in the base, uses three synchronized wheels controlled by two motors, one driving the rotation of all three wheels (translation of the robot) and the other controlling the steering of all three wheels. A third motor in the base controls the rotation of a turret that sits on top of the base, although in these experiments, the turret always pointed in the direction of steering.

Twenty tactile sensors are arranged in two offset rings around the base. The turret contains three sensor systems. A set of 16 sonar sensors, equally spaced in a ring around the turret, provide range data from 6 inches to a maximum of 240 inches. Each sonar cell has a beam width of approximately 22.5 degrees. The turret also contains a ring of 16 active infrared sensors. Using a reflective intensity based system, these sensors give an approximate measurement of range from 0 to 24 inches.

The robot has a two-dimensional structured-light range-finding system, but this sensor is not used in these experiments and is not described further. The robot contains an 80486 processor running at 66 megahertz, a separate microprocessor for the sensors and a separate motor controller. Currently, the processor runs the DOS operating system.

The robot can be controlled either from software running on-board the robot's processor, or from a program running on a host computer via radio modem. In both cases, the programmer uses an identical set of library routines which enable the programmer to both access the sensors and give commands to control the robot. In these experiments, the robot is controlled by giving it velocity mode commands; that is, at each decision step, translation and rotation rates are specified for the wheels, and a rotation rate is specified for the turret. These commands are given via a program running on a Unix workstation.

SIMULATION OF ROBOT

Learning is accomplished off-line using a simulation of the robot. The robot simulation uses the same C language library interface that is used to control the actual robot, so the simulation is the same as the real robot from a programming point of view. However, there are (as expected) significant differences between the robot simulation and the real world robot. Here, we point out some of the significant differences that affect the learning process.

In the robot simulation, both the sonar and the infrared sensors can be set to properly model beam width (the width of the beam is adjustable in the simulation) or they can be set to do simple ray tracing (i.e., an object is only detected if it intersects with the ray drawn along the direction the sensor is pointing). This has the advantage of being significantly faster to simulate. In these experiments, the ray tracing method is used for both the sonar and the infrared. The effect on learning seems to be that the learned behaviors are more cautious since, during learning, the sonar using ray tracing appears very noisy.

In the real robot, the tactile sensors require a certain amount of force to activate them. However, in the simulation, a collision with an object always results in an activation of the tactile sensor. This difference has no direct effect on learning, but it does effect the testing of the final behaviors on the real robot.

In the real world, wheel slippage and irregularities of the floor result in errors in dead reckoning (i.e., determining the robot's position). Although the simulator has parameters to model simple slippage of the wheels, we set the simulation to not model slippage. The resulting simulation has perfect location information from the integrating of its velocity over time. This does not effect these experiments due to the relatively short distances traversed.

Another difference lies in the time lags associated with the real robot being controlled by a remote host. Getting the sensor values from the robot to the host, and the action from the host back to the robot, takes a certain amount of time that is not accurately simulated. This time delay generally results in poorer performance of the behaviors in the real world since the time from the stimulus to the response differs from that experienced in the simulation. In future experiments, the time lag will be more accurately modeled.

DESCRIPTION OF LEARNING TASK

The task to be performed by the robot, for which it must learn a suitable reactive behavior, is to navigate through the room from a given start position to within a certain radius of a given goal location, while avoiding collisions with obstacles randomly placed in the room. Since the robot does not have a global map of the room, this is *not* a path planning problem, but one of local navigation and collision avoidance. The learning task is to evolve a behavior represented as a set of stimulus-response rules that map current sensor state into velocity mode commands for the robot to execute. The decision rate is approximately 1 hertz. The robot is given a limited amount of time to reach the goal.

For this task, the learning algorithm is not directly presented with the 52 individual sensor readings or the integrated robot position, but the algorithm is instead given the following *virtual* sensor information:

Forward_sonar: The minimum value returned by the three forward facing sonars.

Rear_sonar: The minimum value of the three rear facing sonars.

Left_sonar: The minimum value returned by the five left facing sonars.

Right_sonar: The minimum value returned by the five right facing sonars.

The infrared is handled is an identical fashion, with four sensors defined, *forward_ir, rear_ir, left_ir,* and *right_ir*. In addition, the following virtual sensors are defined for the learning algorithm to use:

Time: The current decision time step.

Speed: The current translation rate of the robot.

Range: The range in inches to the goal position.

Bearing: The relative bearing in degrees to the goal position.

Given these sensors, the resulting behavior must, at each decision step, produce two actions: a translation rate for the robot which is between -1 and 5 inches per second, and a turning rate between -40 and 40 degrees per second.

For these experiments, the starting position and the goal position do not change. However, with each trial presented to the robot during learning, the obstacles are placed in different positions. There are two sizes of obstacles placed in the room, one slightly smaller than the diameter of the robot, and the other larger than the diameter of the robot.

RESULTS

For the experiment reported here, the simulated 24 by 30 foot room contained between 5 and 10 obstacles, each obstacle randomly chosen to be either 1.5 or 2.5 feet on a side. The obstacles were placed somewhat randomly: Five slightly overlapping regions in the room were defined roughly corresponding to the North-East, South-East, North-West and South-West sections of the room, with another region defined for the center of the room. Each of the five regions had either one or two of the obstacles placed randomly within its boundaries. This arrangement guarantees that a solution exists, yet forces the robot to have to navigate around several boxes. Each trial begins with a different configuration of the obstacles in the room. The robot was given 80 decision time steps to cross the room to the goal position.

The initial *heterogeneous population* [3] consisted of a variety of rule sets from different sources. This included a combination of manually (human) generated rules sets and automatically generated variants of those rules. The best of these initial rule sets could successfully reach the goal 72.5 percent of the time in simulation, while the worst of them would never reach the goal. The population size was 50 rule sets. Each rule set was evaluated on 20 trials to determine its fitness, which was defined to be the average of the performance measure over the 20 trials. The system was run for 50 generations.

Every five generations, the best 5 rules sets (based on the average performance measure) were tested on 50 random trials to see which could complete the task the greatest number of times. The best performing rule set was then evaluated another 200 times and this value is plotted in Figure 2. As seen in this figure, the best performance behavior could complete the task 93.5 percent of the time in the simulated environment.

After the experiment was completed, the best rule set generated during the experiment (obtained in generation 40) was tested on the real robot. In 15 runs, the robot succeeded 14 times, a performance level of 93.3 percent.

FUTURE WORK

Future work will continue examining the process of building robotic systems through evolution. We want to know how multiple behaviors that will be required for a higher level task interact, and how mutiple behaviors can be evolved

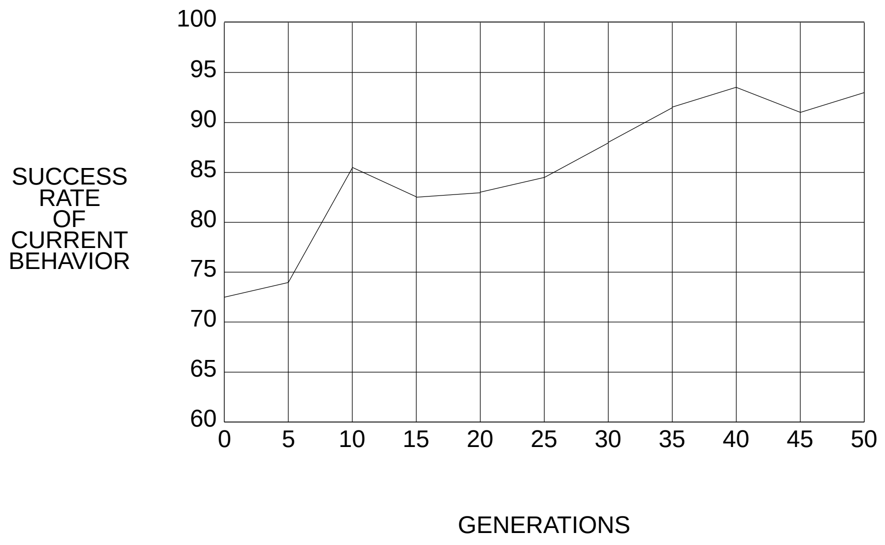

Fig. 2. Learning curve.

simultaneously. We are also examining ways to bias the learning both with initial rule sets, and by modifying the rule sets during evolution through human interaction.

REFERENCES

1. Grefenstette, J. J. and C. L. Ramsey, An approach to anytime learning, *Proceedings of the Ninth International Conference on Machine Learning, 1992,* (pp 189-195), D. Sleeman and P. Edwards (eds.), San Mateo, CA: Morgan Kaufmann.

2. Ramsey, C. L., A. C. Schultz and J. J. Grefenstette, Simulation-assisted learning by competition: Effects of noise differences between training model and target environment. *Proceedings of the Seventh International Conference on Machine Learning, 1990,* (pp 211-215). Austin, TX: Morgan Kaufmann.

3. Schultz, A. C. and J. J. Grefenstette, Improving tactical plans with genetic algorithms. *Proceedings of IEEE Conference on Tools for AI, 1990,* (pp 328-334). Washington, DC: IEEE.

4. Gordon, D. F., An enhancer for reactive plans. *Proceedings of the Eighth International Machine Learning Workshop, 1991,* (pp 505-508). Evanston, IL: Morgan Kaufmann.

5. Schultz, A. C., Using a genetic algorithm to learn strategies for collision avoidance and local navigation. *Seventh International Symposium on Unmanned, Untethered, Submersible Technology, 1991,* (pp 213-225). Durham, NH.

6. Grefenstette, J. J., C. L. Ramsey and A. C. Schultz, Learning sequential decision rules using simulation models and competition. *Machine Learning 5(4), 1990,* 355-381.

7. Grefenstette, J. J., Lamarckian learning in multi-agent environments. *Proceedings of the Fourth International Conference of Genetic Algorithms, 1991,* (pp 303-310). San Diego, CA: Morgan Kaufmann.