IMPLEMENTATION OF GENETIC ALGORITHMS IN SELF-LEARNING MOBILE ROBOTS

Ionut Resceanu[1], **Ofelia Tindea**[2]

[1]*University of Craiova Romania, Faculty of Automation, Computers and Electronics, Department of Mechatronics,*
[2]*S.C. IT SIX Global Services, Craiova, Romania*

Abstract: Applying learning algorithms such as machine learning and reinforcement learning to real physical robots is an area of active research in embodied intelligence. Autonomous, online learning robots possess the ability to operate in complex, dynamic environments through training and instruction to improve the robots' connection between its perception and action. In this article, it is presented an implementation of an autonomous, learning robot that performs exploration of the environment and learns how to avoid obstacle effectively.

Keywords: mobile robot, genetic algorithms, learning, path-planning, control algorithms

1. INTRODUCTION

Autonomous navigation allows robots to plan their path without the need for human intervention. The pathplanning problem has been shown to be difficult, thus this problem is often solved using heuristic optimization methods such as genetic algorithms. An important part of the genetic algorithm solution is the structure of the genotype that represents paths in the navigation environment. The genotype must represent a valid path, but still be simple to process by the genetic algorithm in order to reduce computational requirements. Unfortunately, many contemporary genetic path-planning algorithms use complex structures that require a significant amount of processing, which can affect the real-time response of the robot.

2. OBJECTIVES

As the robot interacts with the environment certain properties of this environment are sensed with the robot's sensors. What makes the robot learning problem more diffcult than other non-robot machine learning problems is the fact that there are a number of additional unknowns the precise states of the robot and the world are unknown and estimates have to be used.

We need to determine the objectives of the learning process. The most precise way to describe the objectives of the robot is to determine sensor states that are to be achieved and sensor states that are to be avoided. This is a description of goals at the level of robot hardware and may not be suitable to describe all conceivable goals for instance some goals may only be describable by multiple sensor states including past sensor readings. However when it is applicable it provides a precise and deterministic way of encoding what the robot is supposed to achieve. We measure the success of any learning strategy in terms of these goals. This is important for the detrmining how the performance of robots depends on the task description. Success is measured in terms of specified goals, not in terms of elegance of solution or performance related criteria.

Our goal is to make the robot perform a specifed task in its environment in as reliable and efcient a manner as possible. Our means is to select the control function $f : S \rightarrow A$ accordingly (S is the current perceived state and A the action chosen).

There are at least two choices for selecting f. The simplest way is to pre-program a fixed strategy that maps perception to action a "hardwired" control program. This works satisfactorily for many basic robotics applications. However, as the complexity of the task or of the environment increases the problem of perceptual discrepancy also increases. This is the problem that the human designer inevitably has a very diferent perception of the world compared with the robot's view, a fact that makes him the less able to determine a priori which of the sensor signals will be relevant for a particular task and which won't the more complex a task becomes. Eventually this discrepancy between human and robot perception can only be resolved through experiments, i.e. trial and error.

The robots should able to learn fundamental motor skills such as obstacle avoidance by associating perceptions with motor. To make a robot with two whiskers avoid convex obstacles a simple hard wired control function of move away from the side where a whisker touches in many cases would sufice.

However because of their learning capability these robots would be able to escape from dead ends even though this situation had never been anticipated during the controller design process (perceptual discrepancy problem) , but they quickly learne to keep turning in one direction whenever any whisker touched.

Reinforcement learning is often a considerable abstraction of the problem of goal-directed learning from interaction. The use of a reward signal to formalize the idea of a goal is one of the most distinctive features of reinforcement learning.

A state signal that succeeds in retaining all relevant information that summarizes past sensations compactly while retaining all relevant information is said to be Markov. In other words, the state transitions have to be independent of the path of the agent until this point and only depends on the current state.

3. ROBOT LEARNING

The robot learning problem is to design a robot so that it improves its performance through experience. To be precise, we must specify what performance and what experience are. Suppose the robot's performance is to be evaluated in terms of its ability to achieve some set of goals G. More precisely, suppose each goal is of the form $X \rightarrow Y : R$ where X and Y are both conditions representing a set of possible states and where R is some real valued reward. The goal $X \rightarrow Y : R$ is interpreted as if the robot finds itself in a state satisfying condition X then the goal of reaching a state satisfying condition Y becomes active, for which a reward R is received. For example_ the goal of recharging the battery when it is low can be represented in this way, by setting X to the sensory input "battery level is low", Y to the sensory input robot senses that it is electrically connected to the battery recharger and R to 100. Given a set of such goals we can define a quantitative measure of robot performance such as the proportion of times that the robot successfully achieves condition Y given that condition X has been encountered, or the sum of the rewards it receives over time. If we wish, we might further elaborate our measure to include the cost or delay of the actions leading from condition X to condition Y.

Given this definition of robot performance relative to some set of goals G we can say that the robot learning problem is to improve robot performance through experience. Thus robot learning is also relative to the particular goals and performance measure. A robot learning algorithm that is successful relative to one set of goals might be unsuccessful with respect to another.

Of course we are most interested in general purpose learning algorithms that enable the robot to become increasingly successful with respect to a wide variety of goal sets.

3.1 What and How to Learn

What and how should we design robots to learn. Two important dimensions along which approaches vary are the exact functions to be learned and the nature of the training information available. Here we consider a few possible learning approaches, then summarise some of the more significant dimensions of the space of possible approaches.

The most direct way to attack the robot learning problem is to learn the control function f directly from training examples corresponding to input-output pairs of f. Recall that f is a function of the form $f : S \rightarrow A$ where S is the perceived state and A is the chosen control action

In some cases, training examples of the function f might not be directly available. Consider for example a robot with no human trainer. with only the ability to determine when the goals in its set G are satisfied and what reward is associated with achieving that goal. For example, in a navigation task in which an initially invisible goal location is to be reached and in which the robot cannot exploit any gradients present in the environment for navigation, a sequence of many actions is needed before the task is accomplished. However if it has no external trainer to suggest the correct action as each intermediate state, its only training information will be the delayed reward it eventually achieves when the goal is satisfied. In this case it is not possible to learn the function f directly because no input-output pairs of f are available. An alternative approach that has been used successfully in this case is to learn a diferent.

3.2 Available Training Information

As is clear from the above discussion, the type of training information available has a strong impact on the choice of learning method. The nature of the training data available can be understood along two orthogonal axes: supervised versus unsupervised learning on the one hand and learning by immediate feedback versus learning by delayed feedback on the other hand.

In the general case of supervised learning, training information about values of the target function are presented to the learning mechanism. The backpropagation algorithm for training artificial neural networks is one common technique for supervised learning.

The training information provided in supervised control learning may be examples of the action to be performed or more limited information in the form of a scalar performance related reward, not indicating the correct action itself. The latter is the case in reinforcement learning as described above. Note that in supervised learning, the supervisor providing the training informationmay be either an external trainer or a module within the robot itself, as in situations where the robot assigns rewards to goal states. We refer to the later as self-supervised learning because one component of the robot is acting a a teacher for another.

Unsupervised learning, on the other hand, performs a clustering of incoming information without using input-output pairs for training. While making robots achieve specified goals, implying that supervised learning will be the chosen method, unsupervised learning also has a role to play. For example, unsupervised learning can identify clusters of similar data points, enabling the data to be reprepresented in terms of more orthogonal features. This reduces the efective dimensionality of the data, enabling more concise data representation and supporting more accurate supervised learning.

Learning by immediate reward has been applied to robot control for a number of decades now. Using electronic circuits whose characteristics were altered through operant conditioning instrumental learning.

As we have seen, training feedback for control learning may be delayed e.g. the reward is provided only at the end of a long sequence of actions leading to the goal or immediate, e.g. the reward or desired action is provided at each step in the sequence leading to the goal. In general, it is more dificult to learn from delayed feedback because the system faces the credit assignment problem, how much did each action of the sequence of actions actually contribute towards accomplishing the desired goal.

Some attempts have been made to make real robots learn from delayed rewards, but the number of training examples required is substantially more in this case, leading some researchers to turn to computer simulations.

3.3 Learning Mechanisms

One clear categorization of learning mechanisms in robotics is based on when this signal is received, immediately after an action is performed - immediate reward or only after a sequence of actions has been performed - delayed reward.

If immediate reward is available, learning in robots can be achieved for example through operant conditioning - instrumental learning. For example changing charges in a capacitor, according to learning rule using immediate feedback led to light-seeking behaviour.

Learning from immediate reward is the most efective way of learning. In some cases, however, immediate reward is not available. If the task has been described in such a way that reward is only obtained occasionally i.e. upon reaching a goal state, the learning process becomes very inefective.

Learning by trial and error from performance feedback, i.e. from feedback that evaluates behaviour but does not indicate correct behaviour is the most accessible and most widely used method of robot learning. This learning from scalar feedback, after having executed a particular action is called reinforcement learning.

3.4 Advanced robot learning

The robot learning curriculum identifed some higher level competences in learning robots.

1.In addition to learning the robot control function State – Action, we can have the robot learn a forward prediction model State - Action – NextState. Because this forward prediction model is task-independent knowledge, it can be used to improve the ability of the robot to learn strategies for new goals or tasks. For example, if this learned forward prediction model is perfectly correct and complete, it can be used to compute the control policy for a new goal simply by searching for a sequence of actions that leads to a goal state ,i.e. traditional AI-style planning. More realistically any learned forward prediction model is likely to be only partly correct. Even in this case it can be used to improve the accuracy of learning control policies for new goals. One method for accomplishing this is Explanation_based neural network, in which approximate previously learned forward prediction models have been shown to signi_cantly improve the accuracy of Q learning.

2.Allow previously learned control behaviors to be used as primitive actions for subsequent learning. For example, we have a robot with primitive actions Forward (d) and Turn(t), that learns a control strategy

for the goal exit the room by going into the hallway, and a second control strategy for traverse the hallway. Given a third goal such as navigate from room A to room B, it would be much easier to solve the goal by using these previously learned behaviors as primitive actions. This kind of hierarchical organization ofers one possible route to scaling up robot learning to tasks of more realistic complexity. The purpose of the light-seakingsystem was not primarily to build a light-seeking robot but to investigate learning from delayed rewards.

3.Temporal aspects as well as perceptual ones need to be represented in the controller, for instance by constructing rich internal states from the temporal sequence of perceptions dynamic robot learning. There is a similarity here to early research in computer vision, it became eventually clear how dificult it is to identify an object using a single frame, whilst the task became much simpler through using sequences of images Active vision is a further advancement of this idea, the vision system now being able to control the focus of attention as well.

4.Anticipation and prediction. Anticipation surprise/novelty detection both in temporal sequences and in single perceptions are needed to achieve the necessary sensor acuity_ The more complex task, robot and environment are, the higher will be the demand on conputing resources. One way to tackle this problem is to focus attention even before sensor signals are obtained, i.e. select the signals that are to be processed further_ rather than take all data and analyze afterwards.(Peng,J. and Williams, R. J.,1996)

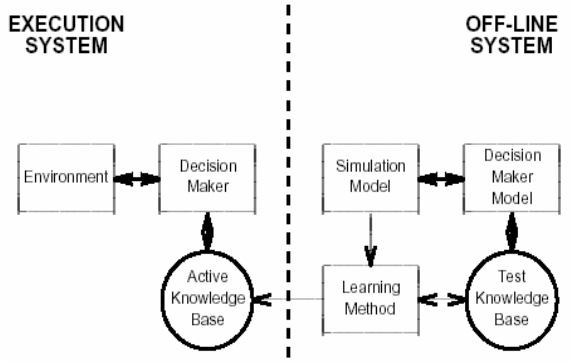

Fig.1. Learning from a Simulation Model.

4. APPLICATIONS OF GENETIC ALGORITHMS IN ROBOT CONTROL

Genetic Algorithms (EA) is a class of algorithms that is inspired by the Darwinian theories of evolution and natural selection proposed by British naturalist Charles Darwin in his publication On the Origin of Species in 1859. There are many parameters that can be adjusted to obtain different emergent behavior of the population as a whole. Genetic algorithms provide us with a new programming paradigm for programming robot controller without deliberate design and tedious parameter tuning on the part of the programmer - the controller is simply evolved over time.
Genetic algorithms belong to the class of stochastic search methods (other stochastic search methods include threshold acceptance, and some forms of branch and bound). Whereas most stochastic search methods operate on a single solution to the problem at hand, genetic algorithms operate on a population of solutions.

To use a genetic algorithm, you must encode solutions to your problem in a structure that can be stored in the computer. This object is a genome (or chromosome). The genetic algorithm creates a population of genomes then applies crossover and mutation to the individuals in the population to generate new individuals. It uses various selection criteria so that it picks the best individuals for mating (and subsequent crossover). The objective function determines how 'good' each individual is.

But there are many ways to modify the basic algorithm, and many parameters that can be 'tweaked'. Basically, if the objective function is defined right, the representation right and the operators right, then variations on the genetic algorithm and its parameters will result in only minor improvements.

There can be used any representation – Figure2 - for the individual genomes in the genetic algorithm: strings of bits, arrays, trees, lists, or any other object. But there must be defined the genetic operators (initialization, mutation, crossover, comparison) for any representation that to be used.

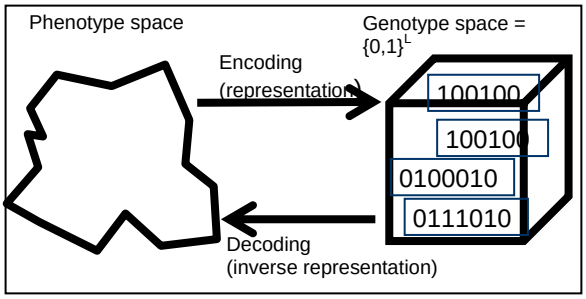

Fig. 2. Representation.

Traditional goal-based robot design has been applied successfully to many domains such as software agents and Web crawlers. Deliberative architectures, for example, are more widespread in software agent technologies rather than physical robot control. The main reason for this is that software agents themselves exists in a symbolic world that is deterministic, and that time is often not a constraint for these agents. This implies that an accurate model of the world can be provided to the robot agents to formulate their plans and actions and that the agents have the luxury of time to formulate these action plans without having to worry about the time

required for reading sensors and controlling actuators. (Smart, D. W., and Kaelbling, L., P, 2000)

However, assuming an accurate representation of the real world is often unrealistic, especially in designing physical robots. One reason for this is because of the dynamism of the real world. For example, the brightness of ambient light is hardly an indication of the time of the day, depending on many factors such as weather conditions, seasons etc. With an almost infinite number of parameters to consider, it would be impossible to accurately represent the real world in a way that will allow robots to function deterministically in the traditional goal-based approach. Another reason is due to the fact that sensors are unable to present a complete picture of the environment - the sensors may be not accurate and precise enough or the sensor channel may be noisy.

The speed of reaction of the robot is also important because it is not realistic to assume that there are not costs associated with using sensors and actuators. In a real robot, it takes time for the sensor to provide the microprocessor with readings and for the microcomputer to move the actuators. Therefore, we cannot afford to have the robot controller perform intensive computation before reacting to the stimuli provided to the robot.

Although reaction time is important, we believe that designing autonomous robots to do the "right thing" is arduous if there is no notion of memory or learning - something that reactive architectures are deficient in. In view of these considerations, we have implemented our robot control using the subsumption architecture to provide the robot with acceptable performance when reacting to its environment and the flexibility to learn and evolve the robot controller over time.

4.1 Obstacles in implementing Genetic Algorithms

There are two main obstacles in implementing genetic algorithms in physical robots. Firstly, the programmer needs to know how to define the fitness function used to evaluate the individuals. This is often a deliberate and complicated task. This task is very similar to supervised learning methodologies used in traditional machine learning and neural networks. Secondly, genetic algorithms often require a fairly large number of individuals and many generations of simulation of this population. This makes running genetic algorithms on robot controllers a very time consuming tasks due to the sheer size of number of simulations that needs to be run on the robots. Currently, genetic algorithms are often applied in simulations and the best evolved controller is downloaded into real robots. However, due to the differences between the ideal world in simulation and the non-ideal world in reality, this method does not provide us with a robust controller that will allow the robot to react optimistically in real world environments.(Brooks, A)

Nonetheless, genetic algorithms provide a way in which complex behaviors can be derived from a group of individual, autonomous robots. It also allows the population as a whole learns the "tricks to life" through evolution. Although it is not clear if genetic algorithms are feasible in the implementation of physical robots, it is still interesting to experiment with these algorithms to derive interesting emergent behavior (if any) from the robots.

4.2 Implementation of a learning robot using Genetic Algorithms

In this section, we describe the implementation of a genetic algorithm on a robot controller in an attempt to evolve the obstacle avoidance behavior. We make use of 8 state FSAs to represent the controller of the robot. The actions Forward, Left, Right and Reverse are encoded (in binary) as 00, 01, 10 and 11 respectively. The robot's genotype (controller) is obtained by first ordering the Old State, Input pairs lexicographically. The robot's genotype is obtained by concatenating the corresponding Next State, Action . The Input value is 1 if the robot's proximity sensor senses an obstacle, 0 otherwise.

Table 1: Parameters for Genetic Algorithm

Parameter	Value
Population Size	5
Generations	10
Probability of Choosing Individual	50%
Steps/Iteration	100
Probability of Mutation	1%
Proportion of Population Mutated	40%
Crossover Points	1
Proportion of Population Crossover	40%

Table 1 shows the list of parameters used in the genetic algorithm that we implemented on the robot .The robot is simulated for 100 exploration time steps for each controller i.e., the Explore AFSM is executed 100 times. At each simulation, the robot is initialized with an initial fitness of 100. If it hits an obstacle, the robot's fitness is decreased by 1. If the robot moves forward, its fitness is increased by 1. Otherwise, its fitness value remains the same. The robot stops after each simulation and will start the next simulation upon user input (via the RCX's PRGM button i.e., we overwrote the function of the PRGM button).

5. EXPERIMENTAL RESULTS

In building the robot, we realize that fine tuning the robot's parameters such as threshold values takes a lot of time and effort and gets increasingly difficult when the robot's controller becomes more complex. We further experimented with genetic algorithms in

an attempt to evolve a robot controller to avoid obstacles. This experiment was not complete but serves as a baseline for using genetic algorithms on our robot implementation.(Smart, D. W., and Kaelbling, L., P, 2000)

We implemented a genetic algorithm on top of the the simple version of the robot controller. The input to the controller is a binary value that tells if there is an obstacle in front of the robot. We overwrite the RCX's PRGM button such that the robot will stop after each simulation and this button will have to be pushed for the robot to start the next simulation (of the next individual). The robot is initialized with a fitness of 100 and this value is decreased when the robot collides with an obstacle. The robot's fitness is incremented only when it moves forward; otherwise, the robot could keep turning in circles and still achieve optimal fitness.

6. CONCLUSION

Our experiments reinforced our knowledge that building a physical robot that operates in the real world environment is very different from simulation. There are a variety of reasons for this phenomenon, including both external and internal (to the robot).

External factors include:
Concurrent tasks: Most robot implementation have some form of concurrent tasks to control the various behaviors of the robots. Too many concurrent tasks proved complex and difficult to get right. Furthermore, since we have only a single processor that implements a time-slicing scheme among the different tasks and the robot is required to react quickly at times, timing became a major issue i.e., insufficient time window to be shared among all the different tasks.

Software abstraction: Software abstraction makes programming a much easier task. The problem with software abstraction is that it tends to abstract away some important timing and synchronization issues in programming the robot. We found that LegOS is a very powerful environment, which allows you to program in the standard ANSI C/C++ and utilize all 32 KB of RAM rather than limiting us to the number of variables as defined by the microcontroller.

REFERENCES

Brooks, A. B. A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation*, Vol. 2, 1: 14-23, March 1986.

Carbonell, J. G., Knoblock, C. A., and Milton, S. PRODIGY: An Integrated Architecture for Planning and Learing. *Technical Report*, CMU-CS-89-189, October 1989.

Chuang-Hue Moh, Research Assignment 4, *6.836 Embodied Intelligence, Massachusetts Institute of Technology*, April 2002.

http://underdog.stanford.edu/tmr/.

LegOS. http://legos.sourceforge.net/.

LeJOS. http://lejos.sourceforge.net/.

Pedersen, M. H., Klitgaard, M. and Thomas, C. Solving the Priority Inversion Problem in legOS. University of Aalborg, UK, May 2000.

Newell, A., and Simon, H. A. Computer science as empirical enquiry: Symbols and search. *Communications of the ACM*, Vol. 19, 3: 113-126, March 1976.

Not-Quite-C.http://www.enteract.com/ dbaum/nqc/.

Peng, J. and Williams, R. J. Technical Note: Incremental Q-learning. *Machine Learning*, 22:283-290, 1996.

Smart, D. W., and Kaelbling, L., P. Making Reinforcement Learning Work on Real-Robots. Research Abstract, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 2000.

Smart, D. W., and Kaelbling, L., P. Practical Reinforcement Learning in Continuous Spaces. In *Proceedings of the Sixteenth International Conference on Machine Learning*, 2000.

The Handy Board. ttp://www.handyboard.com.

The MIT Programmable Brick Project. http://el.www.media.mit.edu/groups/el/-projects/programmable-brick/.