On-line Evolution of Robot Program using a Memoized Function

Worasait Suwannik and Prabhas Chongstitvatana
Department of Computer Engineering
Chulalongkorn University
Phayatai Road, Bangkok, THAILAND 10330
Email: prabhas@chula.ac.th

Abstract
This work proposes a memoized function to speed up on-line evolution of robot programs. On-line evolution is performed on a physical robot. It has an advantage over an off-line method as being robust and does not require the robot model. However, on-line evolution is very time consuming. To validate our proposal, an experiment with visual-reaching tasks is carried out. The result shows that the memoized function can speed up on-line evolution by 23 times and the resulting control program performs robustly.

Keywords:
On-line evolution, Genetic Programming, robot program, memoized function.

1. INTRODUCTION

Classical approaches to robotics require mathematical models in order to plan the robot operations. Because of this requirement, robots that adopt those approaches have to be well engineered and are normally found in a structured environment. In the cases where proper models are difficult to be obtained, the traditional approaches fail to work. To generate a robot controller in such situation, two approaches can be used. The first approach required human intuition to program the robot. A successful example of this approach is Subsumption Architecture proposed by Brooks [1]. The second approach used learning algorithms to automatically synthesize a robot controller. Evolutionary algorithms such as Genetic Algorithms [2] or Genetic Programming [3] allow robots to learn.

Natural evolution is a very slow process. Progress has been made over thousands of generations, which take billions of years. However, with the speed of today's computer, thousands of generations in off-line artificial evolution might take only few hours. Off-line evolution of robot controllers can be done very fast because it occurs in simulation. However, the controller obtained from simulation is not likely to work robustly when transferred to the real robot. This is due to the inaccurate model of the physical world. To overcome this shortcoming, the evolution can be performed with a real robot in the physical world. This is called

on-line evolution. However, it is very time consuming. For example, Floreano and Mondada [4] could evolve an obstacle avoidance behavior on a physical mobile robot in about thirty hours. Chongstitvatana and Polvichai [5] estimates that genetic learning of a robot arm control program will take about two thousands hours.

Artificial evolution can be sped up by using some techniques such as modifying evolutionary algorithms, changing the genetics operators, or reducing the time used in evaluating an individual. Our approach falls into the last category. We reduce the evaluation time by letting the robot learns the effect of its action and used it while evolving a controller for the task.

This paper is organized as follows. Evolutionary robotics is explained in Section 2. Section 3 describes the experiment. Section 4 discusses the result. Section 5 concludes this paper.

2. EVOLUTIONARY ROBOTICS

Natural evolution has created various types of highly fit biological creatures that can survive well in their environments. A group of robotics researchers borrowed this idea in order to create an artificial creature that can successfully perform a task with little human intervention. Evolutionary algorithms have been used successfully to evolve robot programs. Genetic Algorithms (GA) was introduced by Holland [2]. Each individual in a GA population is a fixed-length binary string. Genetic Programming (GP) is a variation of GA introduced by Koza [3]. The major difference between GA and GP is the representation of an individual. Instead of being a fixed-length binary string, an individual in GP is a computer program whose length is varied.

Simulation is useful for study some fundamental problems in evolutionary robotics [6-9]. However, Brooks [10] pointed that when using simulation, we might solve a problem that does not exist in the real world or the solution of the problem cannot be used in the real world. This is because a simulated robot is usually oversimplified. Many aspects of the real world such as noise, uncertainty, mass, friction, inertial forces are ignored.

For these reasons, the evolved controller from simulation does not work robustly when transferred to the real world [11]. Several approaches were proposed to improve the robustness. Some researchers added noise to the simulation to make more accurate simulation [12-14]. Some researchers performed experiments using simulation built from real robot's sensory and actuator data to model the real world more accurately [5,15].

Crossing the simulation-reality gap is not trivial because of two major reasons [13]. First, it is difficult to model any aspect of the real world accurately. Second, it is very difficult to include every aspects of the reality in simulation. Brooks recommended discarding simulation model and using a real robot as its model [1].

On-line evolution of a robot controller is similar to natural evolution in the sense that it occurs in the real world. The tremendous amount of time in on-line evolution leads to several problems [16]. This might be a reason that there are a few attempts to evolve a robot controller on-line. Floreano and Mondada evolved

an obstacle avoidance behavior for a mobile robot [4]. The behavior of neuron network controller converges in about thirty hours. Dittrich and his colleagues evolved a controller for a robot with arbitrary structures [17].

3. MEMOIZED FUNCTION

To evaluate an individual on-line, each robot motion is performed in the physical world. However, many of these motions are repetitive hence their effects are already known. If these effects are stored and reused, then a large number of actual motions can be eliminated. We propose a memoized function to store the effects of robot motions.

The memoized function receives joint angles as its input. It outputs the positions of all joints. Our implementation of the memoized function can hold every possible combination in the joint space. Since each joint of our robot arm has 60 discrete steps, there are a 60^3 or 216,000 entries. However, for larger size of configurations, we do not have to allocate all entries to implement the function. The technique of virtual memory can be used. The memoized function can be implemented with a much smaller physical memory.

4. EXPERIMENT

The aim of this experiment is to evolve a robot program without using any simulation model. The robot learned the visual reaching task on-line using GP. Before the on-line evolution took place, we used simulation to estimate the time that the robot learned the task without using the memoized function. Different sets of genetic parameter were uses in simulation. The best parameter set was used in on-line learning. We compared the estimated time with the exact time spent in on-line evolution using the memoized function. Finally, the robustness of a resulting program from each run is measured.

4.1 The Robot Arm and Its Task

We constructed a three DOF robot arm as our experimental platform as shown in Figure 1. Each joint of the arm is made of servo normally used in a hobby radio controlled airplane or car. The arm can move only in a plane. A CCD camera located above the arm provides a robot vision. The vision system monitors the distance between the tip and the target and checks whether the robot hit any obstacles.

Although the robot looks simple, building an accurate model is not obvious. The real robot as its model gives the following properties:

- Lens distortion. An inexpensive surveillance CCD camera used in this experiment has pretty high distortion. The length of each links varied from angle to angle.
- Camera calibration. By not depending on any mathematical model, there is no need to calibrate the camera to fit with the model. The camera can be placed at any height from the robot-moving plane as long as it can see the arm movement.
- Motor effect. The robot can learn the effect of its motor command such as joint positions during the evolution of control program.

As shown in Figure 2, five instances of the visual-reaching problem similar to those in [14] were created as the representatives of the problem. They are varied in degree of difficulty. A circle in each picture is a target. Black rectangles are obstacles. During the learning period, the robot can freely move from one configuration to another without hitting any obstacles. However, in the testing period, it has to reach the target while avoiding the obstacles.

4.2 The Control Program

GP used primitives in a terminal and a function set to construct a robot program. The terminals set contains robot motion command and sensing primitives. A motion command primitive moves a specific joint one step. A sensing primitive senses if any of the robot's links hit an obstacle or whether the tip is closer to the target. The function set contains basic control flow primitives, which are IF, IF_AND, IF_OR, NOT. The structure of the robot program makes the robot reactive.

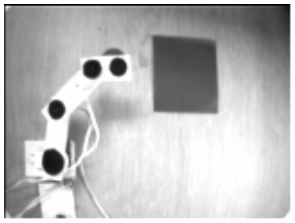

Figure 1. A robot arm seen from its vision

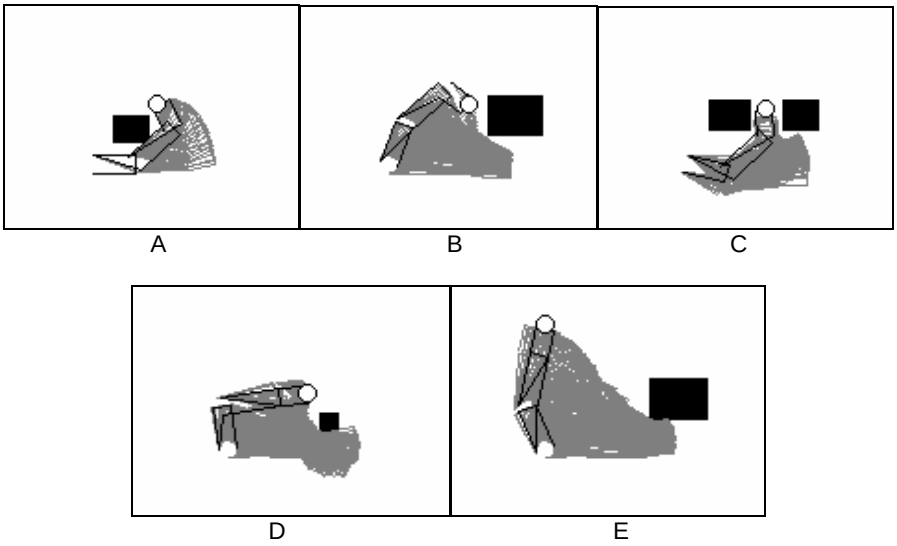

Figure 2. Instances of a target-reaching problem

4.3 Fitness Function

Each robot program is given a limited amount of steps to be executed before its fitness is evaluated. The further the distance from the tip of the arm to the target, the lower the fitness. If the distance between the tip and the target is equal or less than 2 pixels, the robot is successfully perform the task. A fitness function *f* is defined as follows.

$$f(v) = 1000; \quad d(v, tip) \leq 2$$
$$= 1000 - d(v, tip); \quad \text{otherwise}$$

where $d(v1, v2)$ is the distance between point *v1* and *v2*.

4.4 Genetic Parameters

Standard GP is used in our experiment. Before we evolved the robot program on the physical robot, we conduct an experiment in simulation to test the performance of our approach. We found that the population size affected the on-line evolution time. Among tested population size, the size of 200 individuals gives the best estimation result in evolution time.

We made no efforts in optimizing other genetic parameters. The parameters are similar to those in [14] except that the evolution is continued until no progress has been made during 10 consecutive generations. If the evolution stops without a solution, it will be rerun with the memoized function that has already been filled.

5. RESULT

As shown in Table 1, a physical robot can learn the task on-line in less than one hour on average when using the memoized function. We compared the performance of our approach with normal on-line evolution. Since it took a considerable amount of time for a robot to learn the task on-line without using the memoized function, learning time is estimated by using simulation. The estimation assumed that each movement of a simulated robot took one second to complete similar to the physical robot. When compared the estimated time with the actual on-line evolution time, the average speed up is about 23 times.

Table 1. Evolution time

Problem	With Memoize (hours)	Without Memoize (hours)	Speed up (times)	Recall (%)	Memory used (%)
A	0.51	5.00	9.64	89.52	0.90
B	0.64	6.33	7.80	85.65	1.13
C	1.19	25.32	20.93	94.82	2.20
D	1.28	132.68	59.09	93.45	2.31
E	0.93	34.81	17.18	81.91	1.79
Average	0.91	40.83	22.93	89.07	1.67

The recall rate is very high. Each recall means the required data is found in memory, which means that the robot does not have to move in the physical

world. Less than 3% of the memory allocated for the memoized function were used. The usage of the memory depends on the time it takes to learn the task. The first generation filled more entries compared to later generations. This is because the offspring is likely to be in the same configurations as its parents is. The new configurations implies the exploration of the offspring.

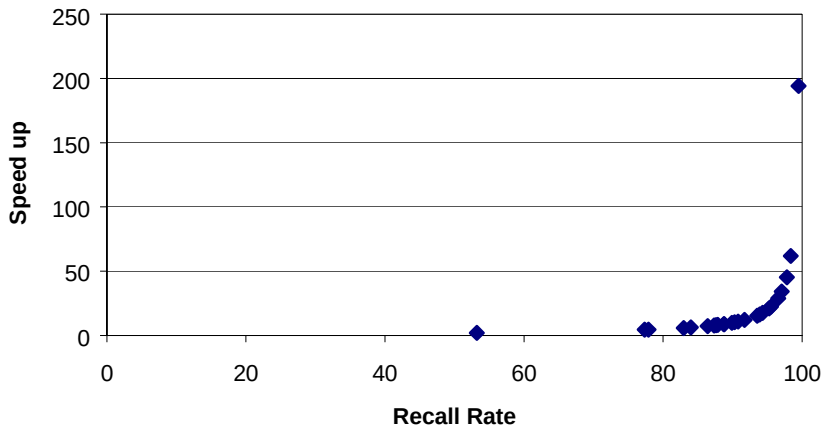

Figure 3. Recall rate versus speed up

As show in Figure 3, the percentage of recall and speed up are exponentially related. Starting from zero recall, the speed up is equal to one (i.e., no speed up.) The speed up grows much larger when the recall rate is above 95%. A controller for a more complex task might be able to evolve on-line in a reasonable amount of time if the recall rate is very high.

The robustness of a robot program is the percentage of times the robot can successfully perform the task. We compared the robustness of a program evolved off-line (i.e., in simulation) and on-line with the memoized function (i.e., in the real world). The result is shown in Table 2. The average robustness of a program evolved on-line is 27% higher than evolved off-line. The robustness depends on the similarity between the environment that is used to evolve robot programs and the environment that the robot programs actually work. The on-line evolution occurred in the actual environment that the robot programs work as opposed to the off-line evolution which used the simulation. Therefore, a program evolved on-line evolution tends to have higher robustness.

We found that the failed program moved to the configuration that is not found in memory many times more than the successful program does. In other words, the failed program moved to the configuration it was not learned during the evolution. We hypothesize that this is due to noise in the system that leads the robot to move to unknown configurations. One way to improve the robustness is to add the same level of noise to the memoized function when evolving a control program.

Table 2. Robustness of an evolve program

Problem	Robustness of a program evolved off-line	Robustness of a program evolved on-line with the memoized function
A	82	98
B	26	76
C	82	82
D	56	60
E	28	92
Average	55	82

6. CONCLUSION

Inaccurate models resulted in fragile behavior of a robot. On-line evolution eliminates the use of any mathematical models of the robot. However, on-line evolution is very time consuming. By using the memoized function, on-line evolution of a visual-reaching task can be speed up by about 23 times. The evolved robot program works robustly in the testing environment. The size of required memory for implementing memoized function grows exponentially with the degree of freedom of the robot. However, the memoized function can be implemented with the technique of virtual memory as it is found that the percent of use is low in our experiment.

To improve the robustness, noise can be added to a memoized function. The memoized function can be replaced by another learning method, which will learn the effect of robot movement. The remaining question is that the speed up is enough to learn more difficult task or with a high DOF robot.

References

1. Brooks R., "New Approaches to Robotics", Science 254, pp. 1227-1232, Sep. 1991.
2. Holland J., Adaptation in Natural and Artificial Systems, University of Michigan Press, 1975.
3. Koza J., Genetic Programming, volume (1), MIT Press, 1992.
4. Floreano D., Mondada F., "Automatic Creation of an Autonomous Agent: Genetic Evolution of a Neural-Network Driven Robot", Proceeding of the 3rd Inter. Conf. on Simulation of Adaptive Behavior, 1994.
5. Chongstitvatana P., Polvichai J., "Learning a visual task by genetic programming", Proceedings of IEEE/RSJ Inter. Conf. on Intelligent Robots and Systems, 1996.
6. Koza J., "Evolution of Subsumption using Genetic Programming", Proceedings of the First European Conference on Artificial Life, pp. 110-119, 1992.
7. Nolfi S., Floreano D., "Learning and Evolution", Autonomous Robots, 1999.
8. Chongstitvatana P., "Using Perturbation to Improve Robustness of Solutions Generated by Genetic Programming for Robot Learning", Journal of Circuits, Systems and Computer, vol. 9, no 1 & 2, pp. 133-143, 1999.

9. Suwannik, W., Chongstitvatana, P., “Improving the Robustness of Evolved Robot Arm Control Programs Generated by Genetic Programming”, Proceedings of Inter. Conf. on Intelligent Technologies, pp. 149-153, December 2000.
10. Brooks R., “Artificial Life and Real Robots”, Proceedings of the First European Conference on Artificial Life, pp. 3-10, 1992.
11. Polvichai J., Chongstitvatana, P., “Visually-guided reaching by genetic programming”, Proceedings of 2nd Asian Conf. on Computer Vision, pp. 329-333, December 1995.
12. Miglino O., Lund H., Nolfi S., “Evolving Mobile Robots in Simulated and Real Environments”, Artificial Life, pp. 417-434, 1995.
13. Jakobi N., Minimal Simulations for Evolutionary Robotics, PhD. thesis, University of Sussex, 1998.
14. Suwannik, W., Chongstitvatana, P., “Improving the Robustness of Evolved Robot Arm Control Programs with Multiple Configurations”, Proceedings of Asian Symposium on Industrial Automation and Robotics, pp. 87-90, May 2001.
15. Lund H., Hallam J., “Evolving Sufficient Robot Controllers”, Proceeding of IEEE Inter. Conf. on Evolutionary Computation, 1996.
16. Mataric M., Cliff D., “Challenges in Evolving Controllers for Physical Robots”, Special Issues of Robotics and Autonomous System, Vol. 19, No. 1, pp. 67-83, October 1996.
17. Dittrich P., Bürgel A., Banzhaf W., “Learning to Move a Robot with Random Morphology”, Proceedings of the First European Workshop on Evolutionary Robotics, pp. 168-178, 1998.