Second Asian Symposium on Industrial Automation and Robotics
BITEC, Bangkok, Thailand
May 17-18, 2001

IMPROVING THE ROBUSTNESS OF EVOLVED ROBOT ARM CONTROL PROGRAMS WITH MULTIPLE CONFIGURATIONS

Worasait SUWANNIK, Prabhas CHONGSTITVATANA
Department of Computer Engineering
Chulalongkorn University
Phyathai Road, Patumwan,
Bangkok, Thailand. 10330
Tel: (662) 218-6983 Fax: (662) 218-6955
E-mail: vac@orange.cp.chula.ac.th, prabhas@chula.ac.th

***Abstract**: This paper proposes a method to improve the robustness of a robot arm control program. The control program is generated in simulation by genetic programming. The robustness is measured in the real world. To improve the robustness, the control program is evolved with multiple robot arm configurations. The result shows that the robustness of a control program is improved by 10% compared to a control program evolved with a single configuration.*
***Keywords:** Robot Programming, Genetic Programming, Robustness.*

1. INTRODUCTION

Planning in classical robotics required exact geometrical model of the robot (Lozano-Peréz et al., 1992). After the model is obtained, the robot used a planner that is created specifically for it to find a plan. There are several advantages of using Evolutionary Robotics over the classical one. First, a geometrical model of the robot is not required. The real robot can use itself as the model (Floreano & Mondada, 1994) (Nordin & Banzhaf, 1997). Second, the robot does not have to be precisely engineered so that the geometrical model of the robot can easily be obtained. Finally, for different types of the robots to do the same task, there is no need to invent a new algorithm for a new robot. Evolutionary approach can used the same program (i.e., a fitness function) for a new type of the robot.

Artificial evolution can be applied to various robotics problems such as planning, creating a reactive neuron controller, and creating a behavior-based program. The evolution starts from a population of purely random solutions (or agents). After that, they are evaluated based on a fitness function. Those random solutions do not perform well in the beginning. However, after being selected and applied the evolutionary operations (e.g., combination and mutation,) the solutions will be improved.

Genetic programming (GP hereafter) applies artificial evolution to generate a computer program (Koza, 1992). We applied it to generate a program for a hand-eye system. The task of the system is the visually guided target-reaching in an area filled with obstacles. By the time consuming nature of evolution, thousands of fitness evaluations have to be done prior to the emergence of the desired behavior. From this reason, a control program is evolved in simulation. Since no simulation is perfect, the control program obtained from simulation frequently fails to work robustly when transferred to the real world.

When using GP to evolve a control program for a target-reaching task, we found that a control program evolved from a simulation with single robot configuration did not work well. Many aspects of the real robot and the real world have not been simulated with enough accuracy. For example, the lens distortion in the vision system had not been included in our simulation. The result is that events that occur in the real world do not have any corresponding events in the simulation. Rather than improving the accuracy of the simulation, we proposed an alternative. The alternative is to evolve a robot program that does not depend on aspects simulated inaccurately. Using multiple configurations to evolve robot programs is one way to achieve that objective.

The organization of the paper is as follow. Section 2 reviews related work. Section 3 explains the robot task. Section 4 defines the robustness of the robot arm control program. Section 5 describes the experiment. Section 6 discussed the result. Section 7 concludes the paper.

2. RELATED WORK

An evolved control program is usually brittle when dealing with an environment that it has never seen during the evolution. In the simulated world, the control program fail to work in an environment that is different from where it was evolved. The mobile robot control program fail to work when obstacles position is moved (Chongstitvatana, 1998) (Chongstitvatana, 1999). A robot arm fail to reach a target when the target position was slightly changed (Suwannik, 2000).

Several Evolutionary Robotics reseachers reported a reality-simulation gap problem. In (Nolfi et al., 1994), an experiment was conducted on a mobile Lego robot. A neural network controller was evolved in a simulation. When transferred to the physical world, the trajectories of the real robot differed significantly from that of the simulated robot. In (Chongstitvatana & Polvichai, 1996), GP was used to evolve a robot arm program in a simulator. Some successful program, when transferred to the real world, failed to reach the target. Obviously, robustness is one of the most important issues the Evolutionary Robotics researchers have to deal with.

By observing the behavior of a real robot, robustness can be improved. Several methods of improving the robustness of a mobile robot control program were proposed. A mobile robot neural network controller evolved in simulation that had an appropriate level of noise is robust (Jakobi et al., 1995). The robustness of mobile robot controllers evolved in a simulation can be improved by training them in multiple trials (Lee et al., 1997) (Jakobi, 1998). Recording sensor and motor responses from a real robot and used the data in the simulation resulted in a robust behavior of a real robot (Lund & Hallam, 1997).

3. ROBOT TASK

This work aims to improve the success rate of robot arm control program in the real world. The robot control program was evolved in simulator using GP. After that, the control program is tested with the real robot. The task of the real robot arm is to reach the target while avoiding the obstacles. It can reach the target if the distance between the tip of the arm and the center of the target is less than or equal to 5 units. The target does not obstruct the arm.

Fig. 1. shows the real robot arm. The arm has three joints, called a shoulder, an elbow, and a wrist respectively. The arm's initial configuration is stretching to the right. The arm can move only in a plane. The joint is made of different models of servos. The joint limit of the first, the second, and the third joint are 164, 180, and 150 degrees respectively. The resolution of the first, the second, and the third joint are 70, 70, and 60 steps respectively. The length of each link was not measured. The robot knows the length of each links from the robot vision. The robot can sense if the robot hit an obstacle and knows the distance between the tip and the target by using its vision. The arm are not allow to move any of its parts out of the visual field.

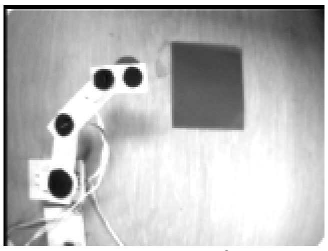
Fig. 1. The robot arm seen from its vision

4. EXPERIMENTS

A robust robot program can perform its task in the real world despite the partial knowledge of the real world in the simulator used to evolved the program. In our experiment, a robot learnt its task in a simulator. The simulator is deliberately built such that it is only an approximation of the real world in three aspects.

- The lens distortion is not included in simulation. The vision system used in the experiment has high distortion due to the close up effect.
- The motion of each joint is not calibrated with the real robot. The accumulated error is as much as 3%.
- The noise in locating the position of each joint due to the vision system is not included in simulation. The vision system locates each joint by recognizing the circular marker and calculates its centroid. The light, which caused uneven reflection in the scene (as seen from Fig. 1), together with the lens distortion resulted in up to ± 2 pixels noise in reporting the position of each joint.

The robustness is measured in the real world as the percentage of time the real robot can successfully reach the target.

Two experiments were conducted. In the first experiment, GP composed a control program that can work with one robot configuration. In the second experiment, we compensated the partial knowledge of the real world by making a control program that did not rely on a robot configuration. With the set up that we used, the major discrepancy between the simulator and the real world is the distortion of the size of the arm caused by the lens. The length of each link varies as the arm moved in the visual field but the simulator had no knowledge about this variation. We decided to choose this aspect for the evolution process to evolve a robot control program that was independent of it. We evolved a program can control three different simulated robot arms to perform the task successfully. Those arms are varied in length. The first arm is the same as that used in the first experiment. The second and the third arm are little shorter and little longer than the first arm respectively.

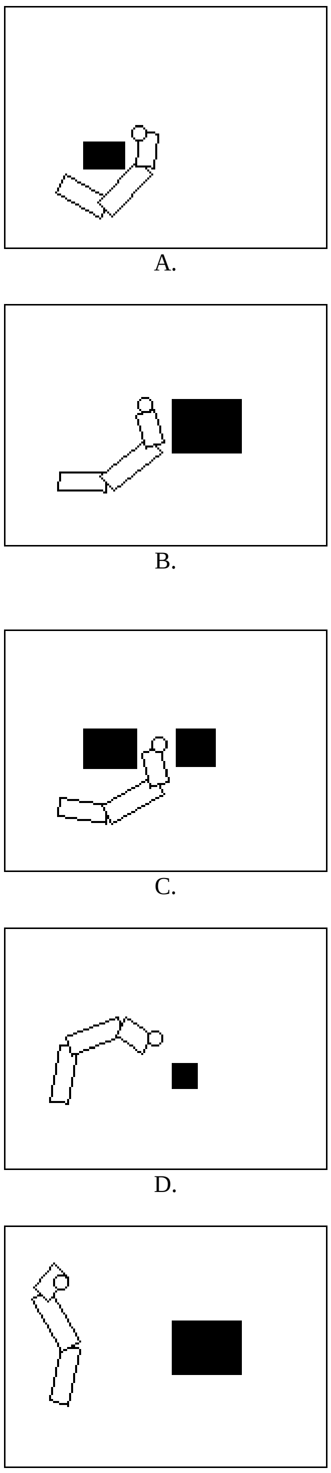

Fig. 2. Five instances of target-reaching task used as a benchmark for robustness testing

As shown in Fig. 2, five instances of the problem were created as the representatives of the problem. They varied in target positions and obstacle positions. A circle in each picture is a target. Black rectangles are obstacles. To make the experiment easier to conduct, the obstacles and the target are not captured from the robot vision. This simplification does not have any effect on the measurement because both types of control programs will be tested with the same benchmark.

4.1 Robot Learning
In both experiments, artificial evolution took place in simulation. For each instance of the task, the evolution was repeated for 10 runs. This is because, due to its probabilistic nature, GP normally gives a different program for each run. Thus, a number of runs are needed to ensure statistically reliable results. The results are averaged over 10 programs generated from GP.

The following subsections describe how we applied GP to the robot learning problem.

4.1.1 Fitness function
Each robot program is given a limited amount of time to be executed in the simulation before its fitness is evaluated. The fitness of a robot program is measured as its ability to go to the target at the end of the run. A fitness function f defined as follow.

$$f(v) = \begin{cases} 100; & d(v, tip) \le 2 \\ 100 - d(v, tip); & otherwise \end{cases}$$

where

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

For the program evolved with multiple configurations, the fitness function is the summation of the fitness function f evaluated from each configuration. The program is accepted as a solution when all three arms can use it to reach the target.

4.1.2 Terminal and function set
The structure of a program obtained from GP is a tree. Each node of a tree in GP population is an executable unit in a terminal or function set. GP composes a program from items in both sets. In our work, each node in the sets returns Boolean value. The terminals set contains robot movement and sensing primitives. Items in the terminal set are listed in Table 1. The function set contains basic control flow primitives. Items in the function set are listed in Table 2.

4.1.3 Genetic parameters
The genetic parameters listed in Table 3 are fixed for all runs. The evolution will stop if the solution is found or it reach the maximum generations. In the latter case, the evolution will be rerun. These genetic parameters are just good enough for GP to learn the task in reasonable amount of time. We did not try to optimize the genetic parameters.

4.2 Measuring the Robustness
After the control program is obtained from simulation, the robustness of robot programs was measured in the real world. They are measure against five instances of the target-reaching problems shown in Fig. 2. Those instances are the same as that the robot had learnt. The different is that a real robot arm is used to run the control program in this measuring phase.

Each program ran on the real robot arm for 10 times. A run that has errors will be discarded. An error was occurred when the robot vision cannot detect all the joints. A run was also discarded when a robot arm cannot 'escape' from an obstacle after it hits the obstacle. A program fails to work when it cannot reach a target in a specified period of time and produces no such errors.

Terminal Name	Operation
Closer	Return true if the tip of the robot is closer to the target. Otherwise, return false.
Sp, Ep, Wp	Rotate a shoulder, elbow, or wrist one step clockwise. Return true if the operation is successful. Otherwise, return false.
Sm,Em,Wm	Rotate a shoulder, elbow, or wrist one step counterclockwise. Return true if the operation is successful. Otherwise, return false.
Farther	Return true if the tip is away from the target. Otherwise, return false.
HitPT	Return true if the last clockwise move results in hitting an obstacle. Otherwise, return false.
HitNT	Return true if the last counterclockwise move results in hitting an obstacle. Otherwise, return false.
OnTarget	Return true if the tip is on the target (i.e., the distance between them is 2 units). Otherwise, return false.
See	Return true if the tip can see the target (i.e., there are no obstacles between them). Otherwise, return false.

Table 1. Terminal set

Function name	Operation
If, IfAnd, IfOr	Basic control flow functions.
Not	Evaluate its child and return the negation.

Table 2. Function set

Name	Value
Population	1,000 programs
The method used for generating the first generation	Grow method with depth limit 4
Crossover rate	80%
Reproduction rate	10%
Mutation rate	10%
Selection method	Tournament selection with tournament size 7
Maximum generation	30 generations

Table 3. Genetic parameters

5. RESULTS

We limited the genetic learning time to 30 generations. For a program that is for one simulated arm, it normally took less than that to evolve a successful controller. In some cases, 30 generations are not enough to find the control program. For a controller that could control three simulated arms to reach a target, it took more effort. Effort also depends on the difficulty of an instance of the problem. Some instances are more difficult to evolve a successful program.

The real world is noisy. The robot arm did not move exactly the same trajectories even it was using the same program. There are errors from several sources. First, the model of the robot is not accurate. The length of arm seen from the vision system is varied due to the lens distortion. The joint step of the robot arm is just the estimation of the real joint step. Second, the robot vision detected different joint position due to variation in lighting and shadows.

Table 4 shows the robustness measure from each map. The robustness value obtained depends on maps. The average robustness of the proposed method is 90%. Training with multiple configurations can improve the robustness by 10%.

6. CONCLUSION

GP can create a robot control program. A control program evolved from simulation is not robust in the real world. The reason is because simulation is not accurate. However, evolving a control program that does not rely on any inaccuracy in simulation can create a robust control program. We identified the main reason that causes the program to fail. To improve the robustness, we evolved a control program that does not rely on the robot configuration.

Instance	Robustness of a control program evolved with one configuration	Robustness of a control program evolved with three configurations
A	86	100
B	92	99
C	81	90
D	84	85
E	54	75

Table 4. Robustness of control programs in various maps

7. REFERENCES

Chongstitvatana P. (1998). “Improving Robustness of Robot Programs Generated by Genetic Programming" in Proceedings of IEEE Asian-Pacific Conference on Circuits and Systems, pp. 523-526.

Chongstitvatana P. (1999). “Using Perturbation to Improve Robustness of Solutions Generated by Genetic Programming for Robot Learning” in Journal of Circuits, Systems, and Computer, volume (9), Nos. 1 & 2, pp. 133-143.

Chongstitvatana P., Polvichai J. (1996) “Learning a Visual Task by Genetic Programming”, in Proceedings of IEEE/RSJ International Conference on Intelligent Robots Systems (IROS-96), pp. 534-540.

Floreano D., Mondada F. (1994). “Automatic Creation of an Autonomous Agent: Genetic Evolution of a Neural-Network Driven Robot” in From Animals to Animats III: Proceedings of the 3rd International Conference on Simulation of Adaptive Behavior, SAB'94. MIT Press/Bradford Books.

Jakobi N. (1998). “Evolving Motion-Tracking Behavior for a Panning Camera Head” in Proceedings of the 5th International Conference on Simulation of Adaptive Behavior, August.

Jakobi N., Husbands P., Harvey I. (1995). “Noise and the Reality Gap: the Use of Simulation in Evolutionary Robotics” in Proceedings of the 3rd European Conference on Artificial Life, pp. 704-720.

Koza J. (1992). Genetic Programming, volume (1).

Lee W., Hallam J., Lund H. (1997). “Applying Genetic Programming to Evolve Behavior Primitives and Arbitrators for Mobile Robots” in Proceedings of the 1997 IEEE International Conference on Evolutionary Computation, pp. 501-506.

Lozano-Peréz T., Jones J., Mazer E., O’Donnell P. (1992). Handey: a robot task planner.

Lund H., Hallam, J. (1997). “Evolving Sufficient Robot Controllers” in Proceedings of the 1997 IEEE International Conference on Evolutionary Computation, pp. 495-499.

Nolfi S., Floreano D., Miglino O., Mondada F. (1994). “How to Evolve Autonomous Robots: Different Approaches” in Evolutionary Robotics, Proceeding of Artificial Life IV, pp. 190-197.

Nordin P., Banzhaf W. (1997). “Real Time Control of a Khepera Robot using Genetic Programming” in Cybernetics and Control, Vol 26 (3), pp. 533-561.

Suwannik W., Chongstitvatana P. (2000). “Improving the Robustness of Evolved Robot Arm Control Programs Generated by Genetic Programming” in Proceedings of International Conference on Intelligent Technologies.