Journal of Circuits, Systems, and Computers

USING PERTURBATION TO IMPROVE ROBUSTNESS OF SOLUTIONS GENERATED BY GENETIC PROGRAMMING FOR ROBOT LEARNING

Prabhas Chongstitvatana
Department of Computer Engineering, Chulalongkorn University
Phayathai Rd., Bangkok 10330, THAILAND
Tel (662) 218 6956, Fax (662) 218 6955
prabhas@chula.ac.th

Received (received date)
Revised (revised date)
Accepted (accepted date)

This paper proposes a method to improve robustness of the robot programs generated by genetic programming. The main idea is to inject perturbation into the simulation during the evolution of the solutions. The resulting robot programs are more robust because they have been evolved to tolerate the changes in their environment. We set out to test this idea using the problem of navigating a mobile robot from a starting point to a target in an unknown cluttered environment. The result of the experiments shows the effectiveness of this scheme. The analysis of the result shows that the robustness depends on the 'experience' that a robot program acquired during evolution. To improve robustness, the size of the set of 'experience' should be increased and/or the amount of reusing the 'experience' should be increased.

1. Introduction

Our main interest is in the automatic generation of robot programs: given a task description and a particular environment, generate a robot program to perform the task. Genetic Programming (GP) [1] can be used to solve this problem. GP can be regarded as a population based search technique which represents candidate solutions as robot programs. The candidate solutions are said to be evolved until the solution is found. GP uses natural-inspired operators such as selection, reproduction, crossover and mutation operated on candidate solutions to perform the search. It searches a large space before it finds a solution. Therefore, for practical reason, the search is performed in a simulation in which the speed of the robot is not a limiting factor.

Even in the simulated world, the robot programs work successfully only in a particular environment which they were evolved on. They may not work even in that environment if it is slightly changed. The robot programs generated by GP are found to be 'brittle' or lack of robustness [2, 3]. That is, they fail to work even when there is a small change in the operational environment. This situation is common when robots work in the real world. The condition for operating a robot program in the real world must be exactly the same as in the simulation, even a small deviation

can lead to failure. The accuracy of the world model is an important factor for success. This can be a problem when we transfer robot programs from simulated world to the real world because simulation cannot model the real world exactly. The problem of transferring the result from the simulated world to the real world has been widely recognised [5, 6, 7]. Improving the robustness of robot programs is essential.

This paper proposes the use of perturbation to improve robustness of the robot programs. The main idea is to inject perturbation into the simulation during evolution of the solutions. The evolution process is carried out such that each individual is evaluated under multiple environments that are variant of the original. The resulting robot programs are more robust because they have been evolved to tolerate the changes in their environment. We set out to test this idea using the problem of navigating a mobile in an unknown cluttered environment. We conduct the experiment with a large number of runs and collect statistical data of robot behaviors. Our analysis is based on the notion of 'trace'. A trace is a record of sequence of robot's primitive actions during execution in an environment. A set of trace in which the robot generates in the training period is called robot's 'experience'. We found two factors affecting robustness: the size of the set of experience and the amount of reusing the experience during execution of the task.

The rest of this paper is organized as follows. Section 2 reviews previous work. Section 3 introduces the problems and the experimental set-up. Section 4 elaborates how to improve robustness. The analysis is done in Section 5. Section 6 concludes the paper and discusses future work.

2. Related Work

In this section we discuss some of previous works that demonstrate the use of GP to generate robot programs, the robustness of those programs and the attempt to promote the robustness.

In Chongstitvatana and Polvichai [9], GP is used to generate robot programs that control a real robot arm reaching for a target by visual feedback from the real world. The robot programs are evolved in the simulation. Then the robot programs are transferred to perform a task in the real world. The result shows that small changes in the real world such as an obstacle is moved from its position or the robot misses a step due to random noise can lead to failure even though that robot program performs successfully in the simulation. In most cases, the evolutionary process capitalizes on the deterministic, repeatable nature of fitness tests. The individual is repeatedly evaluated in a certain environment. The solution only captures particular characteristics of that environment.

Many approaches have been proposed to increase robustness of the evolved program. Reynolds [3] used noise to promote robustness in the obstacle avoidance problem. The noise consists of errors in the robot's input sensors and the output actuators. In his work, Reynolds could not evolve robust controller programs. Later, Reynolds [4] changed from using a variable sensor placement to a fixed sen-

sor placement to reduce the complexity of the problem. Interestingly, it was this modification that produced more robust solutions. He concludes that GP is capable of evolving robust solutions, and that noise discourages brittle solutions. Another approach to cope with changes is co-evolution. Browne [10] used co-evolution technique to evolve vision-based obstacle avoidance agents. Ito, Iba and Kimura [11] found that the redundancy of program was effective for generating robust robot programs in a box moving problem. Prateeptongkum and Chongstitvatana [12] examined robustness improvement of robot programs by function set tuning.

To cope with the real world, many researchers suggest the use of physical robots to learn in the actual environment of the tasks [7, 8, 13, 14, 15]. The robot will learn by trial and error. This approach is suitable for many learning tasks such as learning the association between sensing and effectors. However, the attempt to use GP as the learning method using this approach is likely to take too much time because of the speed limit of a physical robot. The work in [9] shows that for a visual-reaching task, it will take 2,000 hours with their equipment to learn the task. It is possible to reduce the time by running GP in simulation that samples data from the real world [16, 17, 18].

3. Evolving Robot Programs

This section describes the experiment for generating robot programs by GP under the simulation without perturbation. Then in the next section, we introduce the perturbation to improve the robustness of the solutions.

3.1. *Experimental set-up*

Our problem is to find a robot program that control a robot to navigate in a cluttered environment from the starting point to the target. The size of the environment as shown in Figure 1 is 500 × 750 units. There are many obstacles distributed in this environment. The obstacles have several geometrical shapes, each has the average size of 15 × 20 units. The total area of all obstacles is about 20% of the whole area. The starting point and the target are fixed in the position as shown. The obstacles are randomly but carefully placed such that they stay some distance from the starting point and the target to make sure the robot has room to move. The mobile robot has a round shape with the radius 5 units. It can move forward, turn left and turn right. There are sensors for detection the collision with the obstacle, and two sensors to determine a general direction of the target, one on each side of the robot's body.

The terminal set in our experiment is { `forward, turnLeft, turnRight, smellLeft, smellRight` }. Each of them activates the robot's primitive action. All terminals return the value after execution. The forward moves the robot in forward direction by 1 unit, return 1 if it can move successfully and return 0 when it crashes some obstacles or walls. The `turnLeft` and `turnRight` change the robot direction by 22.5° of its previous direction while maintain the position, both of these

return 1. The `smellLeft` and `smellRight` determine whether or not there is the target on its side, return value 1 if there is the target and 0 otherwise. The function set is { `IF-AND, IF-OR, IF-NOT` } with the arity 4, 4 and 3 respectively. This is the basic control flow function.

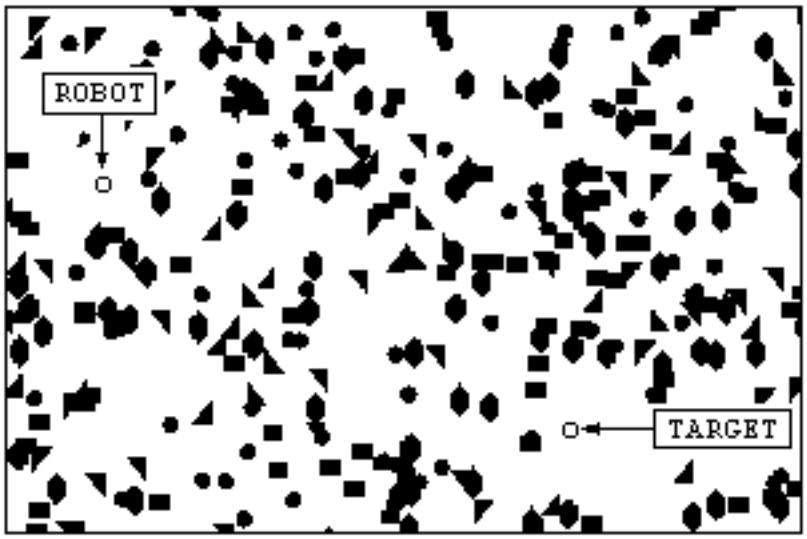

Fig. 1. The environment in our experiment.

The fitness measurement is based on the distance of the robot's final position from the target and the number of its primitive actions. It is measured after a robot program had terminated its execution; when the robot reach the target or it executes more than 5,000 terminals. The fitness function is

$$f = 10,000 \times distance + action \tag{1}$$

where *distance* is Euclidean distance of the final position from the target, *action* is the number of executed terminals. The smaller value of fitness is better. The parameters for GP runs are shown in Table 1. Note that we do not use mutation operator in this experiment. Our crossover operator does not limit the height of the offspring.

3.2. *Robustness testing*

The robustness of a robot program is defined as an ability of the robot program to perform successfully in the unseen environment. In order to measure the robustness of a robot program, we create a number of testing environments by perturbing an original environment. From an original environment, we randomly select the obstacle and move it from its original position by 5 units in a random direction. The number of obstacles that is moved divided by the total number of obstacles is

Table 1. Parameters of GP runs in the experiments.

Population	1,000 programs
Initial size of an individual	110 symbols
Maximum generation	125
Reproduction rate	10%
Crossover rate	90%
Mutation rate	0% (do not use)
Number of repeated run	20 runs

called the percent of disturbance, d.

$$d = \frac{number\ of\ obstacles\ that\ is\ moved}{total\ number\ of\ obstacles} \tag{2}$$

We create 10,000 testing environments and divide them into 10 groups, each has different d, call d_{Test}, that varies from 10% to 100% (each group has 1,000 environments). We then select the best individual from the maximum generation, and evaluate it under these groups of environment. The robustness is measured in each group of environment as the percent of number of environment that the robot program can control robot to the target successfully, denoted as $R(d_{Test})$.

4. Robustness Improvement

In the evolutionary process, an individual is evaluated in one static environment. To improve robustness we introduce perturbation during evolution of the solutions. Perturbation can be introduced into the environment by creating multiple training environments and by changing the percent of disturbance in creating those environments. A number of training environments are created from one original environment using perturbation similar to the way the tested environments are created. To evolve a robot program, each individual is evaluated under multiple environments. The experiments were performed by varying perturbation in two ways:

(i) varying the number of environment during training, keeping the percent of disturbance d_{Train} constant,
(ii) varying the percent of disturbance d_{Train} during training, keeping the number of environment constant.

The fitness of each individual is evaluated by totaling all the fitness value f_i, where f_i is the fitness under the environment i. An individual that works successfully in more environments will have a higher chance to breed to the next-generation.

4.1. *Varying the number of environment*

We use the percent of disturbance $d_{Train} = 10\%$ to create these new environments. We set up 7 different experiments that differ in the number of training environments n, $n = 1$, 5, 10, 15, 20, 30 and 50.

4.2. *Varying the percent of disturbance* d_{Train}

The number of environment n is set to 10. In each experiment, the evolution is performed by evaluating each individual under 10 training environments. The percent of disturbance d_{Train} is varied 10%, 30% and 50%.

4.3. *Result*

We repeat each experiment 20 times with the different initial random population in each run to get reliable statistics. The robustness in each experiment is computed by averaging the robustness of the solution from every run. Figure 2 and 3 plot robustness against d_{Test}. Each line represents how the best robot program performs in the unseen environments. The larger d_{Test} means the more different the environment is from the original. Note that at point $d_{Test} = 0\%$, the robustness of all experiments is 100%. Figure 2 shows the robustness result of the experiment 4.1 and Figure 3 shows the result of the experiment 4.2. In Figure 2, the line Train1 represents the robustness of the robot program that has been evolved without perturbation. Figure 2 shows the robustness increased with the number of training environments, for example at $n = 50$ (Train50), R is higher than $n = 10$ (Train10) for all d_{Test}. Figure 3 shows the robustness increased with d_{Train}, for example at $d_{Train} = 50\%$, R is higher than $d_{Train} = 10\%$ for all d_{Test}. From this result, it is clear that introducing perturbation during the evolution of the solutions can improve the robustness of the solutions.

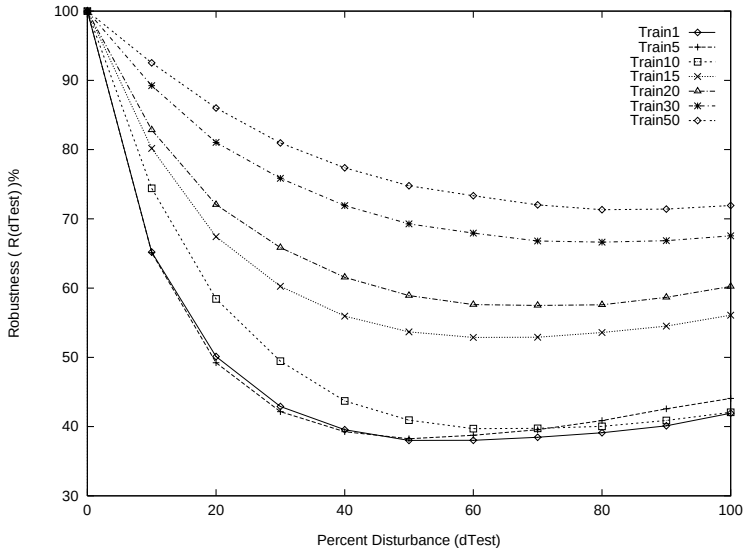

Fig. 2. The robustness of robot programs by varying the number of environments.

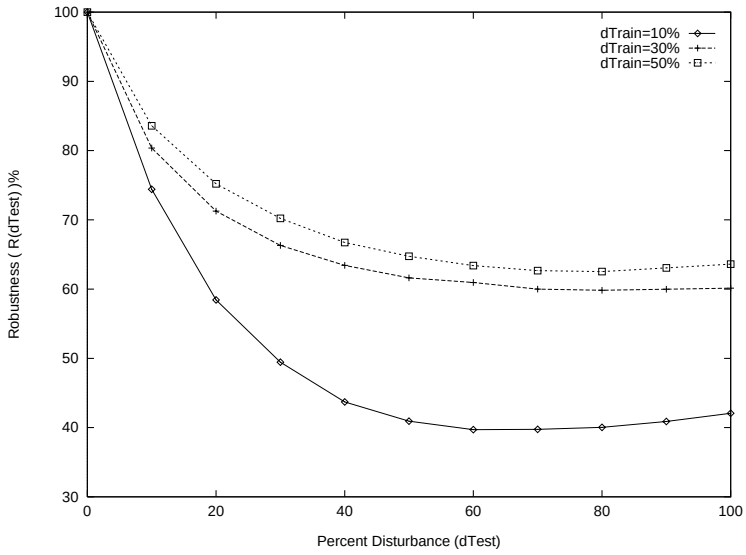

Fig. 3. The robustness of robot programs by varying the percent of disturbance d_{Train}.

5. Analysis of Robustness

In this section, we investigate the cause of improvement of robustness by using the statistics from the experiments. Our analysis is based on the notion of 'trace'.

5.1. *Trace*

An individual (a robot program) can be represented in a tree form (Figure 4). A robot program is evaluated by running the program in the simulation. The evaluation starts from the root node and traverses the tree until reaching the leaf node. This process is repeated until the termination criteria is met. The execution of a robot program produces a number of sequences of actions. We define the sequence of actions occurs from evaluating a robot program once from the root node to the leaf node as a 'trace'. This trace can be regarded as a 'learned' response of a robot program to a particular situation. Evaluating a robot program until it terminates will generate a number of traces.

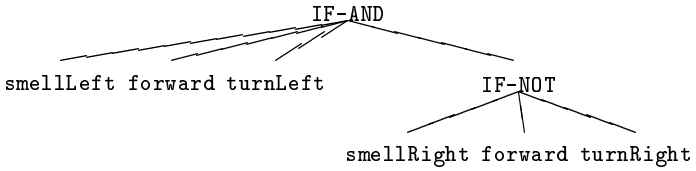

Fig. 4. An robot program represented as a tree.

5.2. *Analysis of traces*

The traces of solutions for all runs in both training and testing phases were collected. We define a set L as a collection traces of robot programs that have been evolved under a number of environments during training phase. A set L can be regarded as a robot's 'experience' or the response to the different situation that robot has learned. Let T_i be the set of all traces of a robot program in the environment i. Let L be the set of all traces T_i in all training environments. We hypothesise that the trace of a robust solution contains a subset of L. By training in many environments, the robot program encodes 'learned' behaviours and hence is able to act properly in a new environment to achieve its goal.

$$L = T_1 \cup T_2 \cup \ldots T_n \tag{3}$$

We define a set A_i, a collection of traces when the robot program is executed in the testing environment i. When executing a robot program in an unseen environment, the robot uses its acquired experience. This notion can be captured with the intersection between L and A_i. The amount of the reuse of the experience of a robot program when executing in i th environment, S_i is defined as

$$S_i = \frac{\mid L \cap A_i \mid}{\mid A_i \mid} \times 100\% \tag{4}$$

where $\mid * \mid$ denotes the cardinality of the set or the number of elements in the set.

The value which we are interested in is the average size of L. We compute the average of S_i for all testing environments. S for an individual that performed successfully is denoted by S_{Succ}, S for the individual that failed is denoted by S_{Fail}. Table 2 shows the result of the experiment 4.1. Table 3 shows the result of the experiment 4.2. It can be seen that S_{All} and $\mid L \mid$ are larger after increasing the perturbation level during training. This fact can be observed in Table 2, S_{All} and $\mid L \mid$ are increased with the number of training environment and in Table 3, S_{All} and $\mid L \mid$ are increased with d_{Train}. The robustness is improved as a result. Also from Table 2 and 3, the successful individual has a higher S than the unsuccessful individual, $S_{Succ} > S_{Fail}$ in all settings. Figure 5 shows the relationship between robustness $R(d_{Test})$ and S_{All}. Figure 6 shows the relationship between S_{All} and $\mid L \mid$. The linear correlation coefficient, r, of each curve is shown in the graphs. The conclusion can be made that $R \propto S_{All}$. A robust solution has a larger set of 'experience' and also reuses them more.

6. Conclusion

This work demonstrates that the robustness of robot programs generated by genetic programming is improved by using perturbation during evolution. Perturbation can be introduced by using multiple training environments and by increasing the perturbation level in each training environment. A solution is robust because it

Table 2. Analysis of robustness when varying the number of training environments.

Exp. Result	at $d_{Test} = 10\%$				at $d_{Test} = 30\%$				$\mid L \mid$
	$R(d_{Test})$	S_{All}	S_{Succ}	S_{Fall}	$R(d_{Test})$	S_{All}	S_{Succ}	S_{Fall}	
Train1	65.19	72.23	83.17	52.61	42.90	55.35	64.19	48.71	24
Train5	65.11	74.77	87.42	52.99	42.15	58.57	69.09	50.88	26
Train10	74.41	82.06	90.77	59.00	49.45	67.75	78.16	57.76	36
Train15	80.17	88.86	93.64	71.26	60.25	80.09	86.29	70.80	38
Train20	82.84	89.89	93.71	72.35	65.85	81.22	85.44	74.21	50
Train30	89.24	95.37	97.09	82.18	75.86	90.86	93.72	82.17	63
Train50	92.52	97.51	98.20	89.59	80.95	94.27	95.85	87.77	96

Table 3. Analysis of robustness when varying the percent of disturbance d_{Train}.

d_{Train}	at $d_{Test} = 10\%$				at $d_{Test} = 30\%$				$\mid L \mid$
	$R(d_{Test})$	S_{All}	S_{Succ}	S_{Fall}	$R(d_{Test})$	S_{All}	S_{Succ}	S_{Fall}	
10%	74.41	82.06	90.77	59.00	49.45	67.75	78.16	57.76	36
30%	80.37	92.63	94.98	83.81	66.31	87.70	90.33	82.55	103
50%	83.58	93.40	95.59	82.93	70.22	88.86	91.69	82.37	90

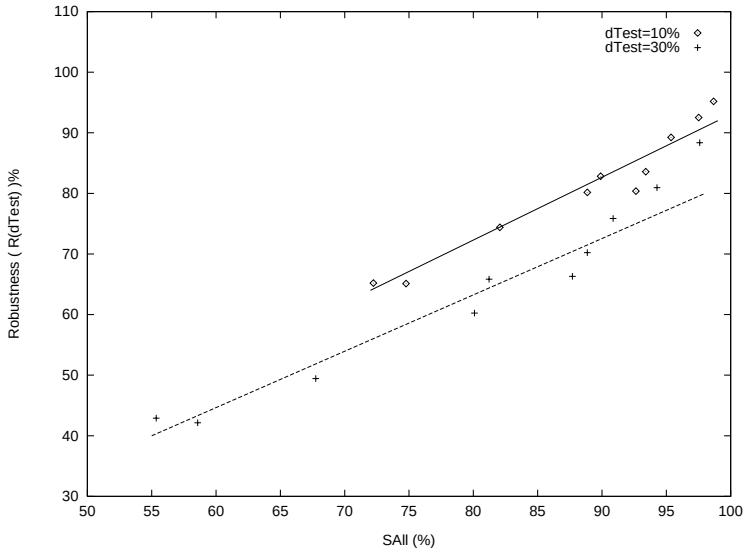

Fig. 5. Relationship of robustness and S_{All} of all experiments.
The correlation coefficients $r = 0.9719$ for $d_{Test} = 10\%$, $r = 0.9721$ for $d_{Test} = 30\%$

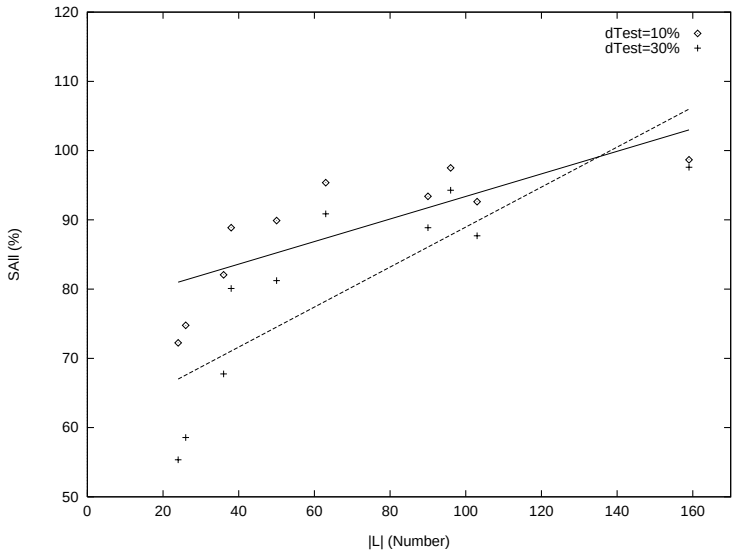

Fig. 6. Relationship of S_{All} and $\mid L \mid$ of all experiments.
The correlation coefficients $r = 0.7962$ for $d_{Test} = 10\%$, $r = 0.8255$ for $d_{Test} = 30\%$

encodes the learned behaviour from the training hence it can act effectively in many environments. Robustness is increased because of the larger size of the robot's experience and the ability of reusing its experience in an unseen environment.

The analysis of robustness gives many insights on the behavior of GP generated robot programs and on finding more robustness improvement techniques. It is noteworthy to observe that many previous work on genetic programming for robot learning are performed with a static environment. The dynamic aspect of the environment is considered a disadvantage that must be dealt with. This work actually exploits this dynamic aspect of the environment and uses it to improve the quality of the solution. Our current activity is concentrated on validating this scheme with the real robot performing in the real world.

Acknowledgements

I would like to thank Roongroj Nopsuwanchai who implemented all the experiments and the review committee whose comments help to improve the presentation of this paper. This research is funded by Thailand Research Fund contract no. RSA/23/2540.

References

1. J. Koza, *Genetic Programming*, Vol. 1, MIT Press, 1992.
2. C. Reynolds, "An Evolved, Vision-based Model of Obstacle Avoidance Behaviour", in *Artificial Life III, Proc. vol. XVI, C. Langdon*, Ed., Addison-Wesley, 1993.

3. C. Reynolds, "Evolution of obstacles avoidance behavior: using noise to promote robust solutions", in K. Kinnear, Ed., *Advances in genetic programming.* MIT Press, 1994, pp. 221-241.
4. C. Reynolds, "Evolution of Corridor Following Behavior in a Noisy World", in *Simulation of Adaptive Behavior*, 1994.
5. R. Brooks, "Artificial Life to actual robots", in *Proc. of the first European Conf. on Artificial Life*, MIT Press, 1991, pp.3-10.
6. T. Chang, S. Kuo and J. Hsu, "A two phase navigation system for mobile robots in dynamic environments", in *Proc. of 1994 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, Vol. 1, pp. 306-313.
7. M. Dorigo, "ALECSYS and the AutonoMouse: Learning to control a real robot by distributed classifier systems", *Machine learning*, vol. 19, 1995, pp. 209-240.
8. M. Dorigo, and M. Colombetti, *Robot Shaping : An Experiment in Behavior Engineering*, MIT Press, 1998.
9. P. Chongstitvatana and J. Polvichai. "Learning a visual task by genetic programming", in *Proc. of IEEE/RSJ Int. Conf. on Intelligent robots and systems (IROS-96)*, Osaka, 1996.
10. D. Browne, "Vision-based Obstacle Avoidance : A Coevolutionary Approach", Bachelor degree thesis in the department of software development, Monash university, Australia, 1996.
11. H. Ito, H. Iba and M. Kimura, "Robustness of robot programs generated by Genetic Programming", in *Proc. of Conf. Genetic Programming 96*, MIT Press, 1996.
12. M. Prateeptongkum and P. Chongstitvatana, "Improving the Robustness of a Genetic Programming Learning", in *Proc. of 3rd Annual National Symposium on Computational Science and Engineering*, Bangkok, 1999, pp. 301-305.
13. M. Mataric and D. Cliff, "Challenges in evolving controllers for physical robots", in *Robotics and autonomous systems*, 19(1):67-83, 1996.
14. M. Olmer, P. Nordin and W. Banzhaf, "Evolving real-time behavioral module for a robot with GP", in *Proc. 6th International Symposium on Robotics And Manufacturing (ISRAM-96)*, Montpellier, France, 1996, in Robotics and Manufacturing, M. Jamshidi, F. Pin, P. Dauchez (Eds.), Asme Press, New York, 1996, pp. 675 -680.
15. R. Arkin, *Behavior-based Robotics*, MIT Press, 1998.
16. O. Miglino, H. Lund, and S. Nolfi, "Evolving mobile robots in simulated and real environments", in *Artificial Life 2(4)*, 1996.
17. W. Lee, J. Hallam, and H. Lund, "Applying genetic programming to evolve behavior primitives and arbitrators for mobile robots", in *Proc. of IEEE Int. Conf. on Evolutionary Computation, 1997, pp. 501-506.*
18. P. Nordin, and W. Banzhaf, "An on-line method to evolve behavior and to control a miniature robot in real time with genetic programming", in *Adaptive Behavior*, 5(2) : 107-140, 1997.