รายงานวิจัยฉบับสมบูรณ์

โครงการ ผลเฉลยที่ทนทานโดยการโปรแกรมพันธุกรรมสำหรับการเรียนรู้ของหุ่นยนต์

โดย ประภาส จงสถิตย์วัฒนา

มีนาคม 2548

สัญญาเลขที่ RSA 4080023

รายงานวิจัยฉบับสมบูรณ์

โครงการ ผลเฉลยที่ทนทานโดยการโปรแกรมพันธุกรรมสำหรับการเรียนรู้ของหุ่นยนต์

ผู้วิจัย สังกัด

ประภาส จงสถิตย์วัฒนา จุฬาลงกรณ์มหาวิทยาลัย

สนับสนุนโดยสำนักงานกองทุนสนับสนุนการวิจัย
(ความเห็นในรายงานนี้เป็นของผู้วิจัย สกว. ไม่จำเป็นต้องเห็นด้วยเสมอไป)

TRF Basic Research
Final Report

Contract number RSA 4080023

1998-2000

Robust Solutions by Genetic Programming for Robot Learning

Prabhas Chongstitvatana
Department of Computer Engineering
Faculty of Engineering
Chulalongkorn University

Project summary

Robotics undoubtedly has an important application in industry. The robot in the future must be adaptable to be able to perform many novel tasks. Robot learning is the key issue in providing that adaptability. This research project aim to understand how a robot can perform many tasks in the real world. The aim of this research is to develop methods to generate robust solutions for the problems in robot learning. The technique that we concentrate on is genetic programming. The problem of robot learning is how to make robots to do tasks without explicitly programming them. It has been intensively studied for the last decade. We view robot learning as a special case of the general problem of "machine learning". Machine learning is a sub-field of artificial intelligence (AI), whose ultimate goal is to replace explicit programming by "teaching". Genetic programming is a unsupervised learning method. It is a general-purpose search algorithm that uses principles inspired by population genetics to evolve solutions to problems. Most of the work in genetic programming research use the simulated world to perform learning and there is problems of transferring the result from the simulated world to the real world. The solution is *brittle*, i.e. the condition for the actual run of a robot program in the real world must be exactly the same as in the simulation, even a small deviation can lead to failure. We attack the problem of how to use genetic programming to generate robust solutions for robot learning from two perspectives : on-line learning and introducing perturbation into simulation. The previous works mostly rely on accurate world model, disallow uncertainty. Our work is different that the system learn the world model and allow uncertainty in the system. An appropriate application of robot learning concept can yield a higher performance in automation which might prove to be very cost effective for many applications in industries.

Keywords : robotics, robot learning, genetic programming, evolution computing, robust solution.

โครงการ

ผลเฉลยที่ทนทานโดยการโปรแกรมพันธุกรรมสำหรับการเรียนรู้ของหุ่นยนต์

โครงการโดยสรุป

ศาสตร์ของหุ่นยนต์มีการประยุกต์ในอุตสาหกรรม หุ่นยนต์ในอนาคตต้องสามารถปรับตัวเองได้เพื่อที่จะทำงานแบบใหม่ ๆ การเรียนรู้ของหุ่นยนต์เป็นหัวใจในการอำนวยให้เกิดการปรับตัวนั้น โครงการวิจัยนี้เป็นงานต่อเนื่องจากความพยามที่จะเข้าใจว่าหุ่นยนต์จะทำงานหลาย ๆ อย่างในโลกจริงได้อย่างไร วัตถุประสงค์ของการวิจัยนี้คือการพัฒนาวิธีการสังเคราะห์ผลเฉลยที่ทนทานของปัญหาเกี่ยวกับการเรียนรู้ของหุ่นยนต์

วิธีการที่เราสนใจโดยเฉพาะคือการโปรแกรมพันธุกรรม ปัญหาของการเรียนรู้ของหุ่นยนต์คือจะทำให้หุ่นยนต์ทำงานได้อย่างไรโดยไม่ใช้การโปรแกรม ปัญหานี้มีการศึกษาอย่างละเอียดในทศวรรษที่ผ่านมา เรามองการเรียนรู้ของหุ่นยนต์ว่าเป็นกรณีเฉพาะของปัญหาทั่วไปของการเรียนรู้ของแมชชีน การเรียนรู้ของแมชชีนเป็นศาสตร์ย่อยของศาสตร์ปัญญาประดิษฐ์ซึ่งมีเป้าหมายสูงสุดคือ การแทนที่การโปรแกรมโดยการสอนการ โปรแกรมพันธุกรรมเป็นวิธีการเรียนรู้โดยไม่มีครู เป็นขั้นตอนวิธีการค้นหาเอนกประสงค์ที่ใช้หลักการของพันธุกรรมประชากร เพื่อที่จะวิวัฒน์ผลเฉลยของปัญหา งานวิจัยส่วนใหญ่ในการโปรแกรมพันธุกรรมใช้โลกจำลองในการเรียนรู้มีปัญหาในการนำเอาผลลัพธ์ที่ได้จากโลกจำลองนี้ไปใช้ในโลกจริง กล่าวคือผลเฉลยมักจะ “เปราะบาง” นั่นคือ สภาวะเงื่อนไขในการทำงานในโลกจริง ต้องเหมือนในโลกจำลองอย่างเคร่งครัด แม้แต่ความคลาดเคลื่อนเล็กน้อยก็จะนำไปสู่ความล้มเหลวได้ เราศึกษาปัญหานี้จาก 2 มุมมอง : การเรียนแบบออนไลน์ และการใส่สัญญาณรบกวนไปในการจำลองแบบงานก่อน ๆ ต้องพึ่งพาแบบจำลองของโลกที่แน่นอน และห้ามมีความไม่แน่นอน งานวิจัยมีความแตกต่างจากงานอื่น ๆ ตรงที่ว่าระบบจะเรียนแบบจำลองของโลกเอง และยินยอมให้มีความไม่ แน่นอนได้ การประยุกต์วิธีการเรียนรู้ของหุ่นยนต์อย่างเหมาะสมสามารถทำให้เกิดประสิทธิภาพสูงในระบบอัตโนมัติ ซึ่งอาจมีผลตอบแทนที่สูงในการใช้งานอุตสาหกรรม

คำหลัก: ศาสตรหุ่นยนต์ การโปรแกรมพันธุกรรม การคำนวณเชิงวิวัฒนาการ ผลเฉลยที่ทนทาน

Research output

Prabhas Chongstitvatana

Contract number RSA 4080023

Title : Robust solutions for robot learning by genetic programming.

1. Publication in journal

   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.

2. Printed book --- none
3. Patent ---- none

4. Publication in the conference proceeding

[1] Chongstitvatana, P., "Improving Robustness of Robot Programs Generated by Genetic Programming for Dynamic Environments", IEEE Asia Pacific Conference on Circuits and Systems, 1998, pp.523-526.
[2] Manovit, C., Aporntewan, C., and Chongstitvatana, P., "Synthesis of Synchronous Sequential Logic Circuits from Partial Input/Output Sequences", Int. Conf. on Evolvable Systems, Lausanne, Switzerland, 1998, pp. 98-105.
[3] Jassadapakorn, C. and Chongstitvatana, P., "Reduction of Computational Effort in Genetic Programming by Subroutines", National Computer Science and Engineering Conference, Bangkok, Thailand, 1998.
[4] Chongstitvatana, P. and Aporntewan, C., "Improving Correctness of Finite-State Machine Synthesis from Multiple Partial Input/Output Sequences", Proc. of NASA/DoD Workshop of Evolvable Hardware, Pasadena, California, July 19 - 21, 1999, pp. 262-266
[5] Nopsuwanchai, R. and Chongstitvatana, P., "Analysis of Robustness of Robot Programs Generated by Genetic Programming", Asian Symposium on Industrial Automation and Robotics, Bangkok, 1999, pp. 211-215
[6] Prateeptongkum, M. and Chongstitvatana, P., "Improving the Robustness of a Genetic Programming Learning", Annual National Symposium on Computational Science and Engineering, Bangkok, 1999, pp. 301-305.
[7] Aporntewan, C., and Chongstitvatana, P., "An on-line evolvable hardware for learning finite-state machine", Int. Conf. on Intelligent Technologies, Bangkok, December 13-15, 2000, pp.125-134.
[8] Sukkarnjanonth, T., and Chongstitvatana, P., "Using non-determinism to improve robustness of robot programs generated by genetic programming", National Computer Science and Engineering Conference, Bangkok, November 16-17, 2000, pp.81-85.

5 Utilization of the research ---- none

6 Development of new researchers

6.1 The number of student who has graduated : 4 Master students

6.2 Thesis

[1] Roongroj Nopsuwanchai. Improving evolutionary process in the genetic programming for generating a robust solution to the robot navigation problem. MSc thesis. Chulalongkorn University. 1998.

[2] Chaiwat Jassadapakorn. Reduction of computational effort in robot learning by genetic programming. MSc thesis. Chulalongkorn University. 1998.

[3] Maria Prateeptongkum. Improvement of robustness of a genetic programming learning method by parameters tuning for the robot navigation problem. MSc thesis. Chulalongkorn University. 1998

[4] Chatchawit Aporntewan. A mimetic evolvable hardware for sequential circuits. MEng thesis. Chulalongkorn University. 1999.

7 Other output

[1] Give a lecture on "Tutorial on Evolutionary Computation" at National Computer Science and Engineering Conference, Bangkok, Thailand, 19 October 1998.

[2] Give a lecture for the Thailand Computer Olympic team on the subject titled "Genetic Algorithm" at the institute for promoting science and technology teaching (IPST), Bangkok, 28 October 1998.

[3] Give a seminar on "Genetic Algorithms" at Department of Computer Engineering, Chulalongkorn university, in occasion of 25th anniversary of the department, June 2000.

Note

All published works can be downloaded from my homepage at:

http://www.cp.eng.chula.ac.th/faculty/pjw/

Select Publication list

Robust Solutions by Genetic Programming for Robot Learning

Project investigator

Prabhas Chongstitvatana

1. Motivation

Robotics undoubtedly has an important application in industry. The robot in the future must be adaptable to be able to perform many novel tasks. Robot learning is the key issue in providing that adaptability. This research project is a continuing work in which we aim to understand how a robot can perform many tasks in the real world. Our previous work focused on robot programming (Chongstitvatana, 1992; 1994), we expanded the work to study robot learning (Chongstitvatana and Polvichai, 1996). We applied the genetic programming method as it is a very suitable technique to automatically generate robot programs. We came up with the conclusion that robustness is an important issue if genetic programming is to be used successfully to let a robot learn to perform its task in the real world. We propose to study this issue. We expect to gain knowledge how to improve this learning technique so that the generated solution is more tolerant to perturbation. The success of this project will promote a more active research in this area.

2. Research goal

The aim of this research is to develop methods to generate robust solutions for the problems in robot learning. The technique that we concentrate on is genetic programming.

Keywords : robotics, robot learning, genetic programming, evolution computing, robust solution.

3. Research problems

This section will review previous work and explain some basic concept in our research. The aim is to put genetic programming into a perspective regarding robot learning. Robot learning is a large field by itself therefore it is important to define what kind of robot learning we propose to do. The following sections describe four major formulations in robot learning and the genetic programming method which is the main technique used in this research. We define "robustness" at the end of this section.

3.1 Robot learning

The problem of robot learning is how to make robots to do tasks without explicitly programming them. The problem of robot learning has been intensively studied for the last decade. There are several text books and articles which give excellent reviews on robot learning including Connell and Mahadevan (1993), Franklin et al (1996). We will examine four major formulations of the robot learning problem : inductive concept learning, explanation-based learning, reinforcement learning, and evolutionary learning.

We view robot learning as a special case of the general problem of "machine learning". Machine learning is a subfield of artificial intelligence (AI), whose ultimate goal is to replace explicit programming by "teaching". Machine learning research has studied many different types of learning. A recent text book by Mitchell (1997) provides a comprehensive introduction to the field. There are two types of learning in general : supervised and unsupervised. In supervised learning, the teacher carefully selects examples for the learner, whereas in unsupervised learning, the learner is given little or no feedback in the learning task.

We can distinguish three types of useful knowledge for robots to learn.

- control: learn a sequence of action to achieve a given goal. Learn which action to perform in a given situation.
- environment : learn the model of environment which can be dynamic.
- sensor-effector : learn to model its sensors and effectors.

We will characterise the problem of robot learning as one of learning a "policy" function Π from some set of sensory states S to some set of actions A, this characterisation is similar to Nehmzov and Mitchell (1990). The "policy" can be "stationary", that is the mapping is a time-invariant function or it can be "non-stationary".

Inductive concept learning

This is the most classical method in machine learning. The robot knows that the policy Π it is learning comes from some space of hypotheses H. The robot is provided with a set of training examples E of the target policy. These are drawn from some space of instances I according to some unknown but fixed probability distribution P.

Given

1. A space of hypotheses H, each of which describes a policy function $\Pi : S \rightarrow A$ over some set of states S and actions A.
2. A set of training examples E of the target policy Π^*, sampled using a (fixed but unknown) probability distribution P on the underlying instance space I.

Determine

A policy $\Pi \in H$ that minimises the expected error with respect to the target policy, measured over I using distribution P.

Two of the most well-known algorithms for solving the inductive concept learning are decision trees (Quinlain, 1986) and neural network (McClelland and Rumelhart, 1986). There are many examples of inductive learning applied to robot learning (Pomerleau, 1990; Littman et al, 1992; Connell and Mahadevan, 1993; Dorigo, 1996; Franklin et al, 1996). The principal weakness of the inductive learning method is that it requires an experienced teacher, who can supply a diverse enough set of training examples.

Explanation-based learning

This method studies how domain knowledge about the function being learned can be used to speed up learning (Dejong and Mooney, 1986; Mitchell et al, 1986; O'Sullivan et al, 1995).

Given

1. A space of hypotheses H, each of which describes a policy function $\Pi : S \rightarrow A$ over some set of states S and actions A.
2. A domain theory D describing the target function Π.
3. An "operationality criterion", which constraints the representation of the learned policy.
4. A set of training examples E of the target policy Π *, sampled using a probability distribution P on the underlying instance space I.

Determine

A policy $\Pi \in H$ whose description satisfies the operationality criterion, and is consistent with D and minimises the expected error with respect to the target policy Π*.

Dejong (1995) described an algorithm which learns to control a simulated robot arm. The primary advantage of EBL in robot learning problem is that it provides a way to incorporate previous domain knowledge to speed up the learning process. However the robot needs to be given some domain knowledge.

Reinforcement learning

Reinforcement learning (RL) studies the problem of inducing by trial and error a policy that maximizes a fixed performance measure (Sutton, 1990; Kaelbling, 1993; Franklin et al, 1996). RL is an unsupervised method, examples are not carefully selected by a teacher. Instead, the distribution of examples is influenced by the robot's actions, since the states and rewards experienced by the robot depends on the action it takes.

Given

1. A space of hypotheses H, each of which describes a policy function $\Pi : S \rightarrow A$ over some set of states S and actions A.
2. A reward function $r : S \times A \rightarrow R$.
3. An optimality criterion O that maps any policy Π and reward function r to a value function $V_r^{\Pi} : S \rightarrow R$.

Determine

An optimal policy Π * that results in a maximal value function V* over all other policies.

RL has several nice properties. It does not require supplying the robot with a theory of its domain. It can be used for on-line learning, thus the robot is continually improving its performance as it learns. Mahadevan and Connell (1992) conducted a study of using RL to

train a real robot to do a box-pushing task that performed better than a hand-coded program. A well-known model free RL algorithm is Q-learning (Watkins, 1989). RL method does not require supplying the robot with either trainning examples or any background knowledge beyond a scalar rewarding function. However, learning can be very slow. RL also assumes that the environment can be modeled as a Markov decision process (MDP) (Puterman, 1994). This "memoryless" is clearly false. Many works are extending RL to partially observable MDP (Littman et al, 1992; Russel and Parr, 1995; Koenig and Simmons, 1996).

Evolutionary learning

Evolutionary learning is unsupervised learning method. Majority of evolutionary learning represented by genetic algorithms (Holland, 1975) and genetic programming (Koza, 1992). Evolutionary starts with a population of policies and combines them to produce better policies until an optimal policy is found. The combination is achieved through genetic operators, such as recombination and mutation. Genetic algorithm encoded each policy as bit string whereas genetic programming encoded as a higher level representation, a tree. The fitness function is a measure of the goodness of a given policy.

Given

1. A space of hypotheses H, each of which describes a policy function $\Pi : S \rightarrow A$ over some set of states S and actions A.
2. A fitness function $f : \Pi \rightarrow R$.
3. An encoding E mapping policies Π to a representation.
4. A set of genetic operators $O : H \rightarrow H$ that transform policies Π.

Determine

An optimal policy Π^* that maximises the fitness function f.

Dorigo (1995), Grefenstette and Schultz (1994) used a classifier system (based on genetic algorithms) to train robots. These systems have been used to train mobile robots to avoid obstacles, to navigate around the rooms, and learn goal-seeking behaviours. Davidor (1991) reported the use of genetic algorithms to plan trajectories of a robot arm. Morowitz and Singer (1995) edited a proceeding that reflected a recent thinking about an adaptive systems (including evolutionary systems).

3.2 Genetic programming

Genetic Programming (GP) is a machine learning method derives from genetic algorithms (GA). Genetic algorithms, originated by Holland (1975), are general-purpose search algorithm that use principles inspired by population genetics to evolve solutions to problems. In Genetic programming, genetically breed populations of computer programs are used to solve problems. The individuals in population are compositions of functions and terminals which are suitable for the intended problem domain. An evolutionary process is driven by the measure of fitness of each individual computer program in handling the problem environment.

We will illustrate how GP can be applied to robot learning problems by describing an example from our previous work (Chongstitvatana and Polvichai, 1996). The problem statement is to learn to visually guide a robot hand to reach a target while avoiding

obstacles. The robot system consisted of a 3 degrees of freedom planar arm and a vision system capable of locating the arm, the obstacles and the target in the scene (figure 1). GP is applied to solve this problem by evolving robot programs that can perform the task. A robot program has terminal set composed of commands to move each joint and sensing commands. A set of functions is { IF-AND, IF-OR, IF-NOT}, figure 2 shows an example of a robot program. Steps of GP process are as following :

1. Create an initial population consisted of robot programs that randomly mixed the functions and terminals.
2. Run each program until it terminates.
3. Evaluate each program, the fitness is measured based on how well it performs the task.
4. Select good programs and do genetic operations (recombination and mutation) on them to generate the next generation population.

These steps are repeated until satisfactory solutions are found.

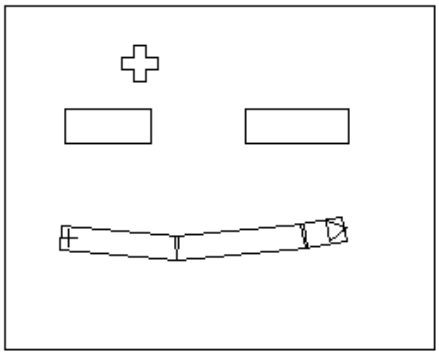


Figure 1. A visual reaching task

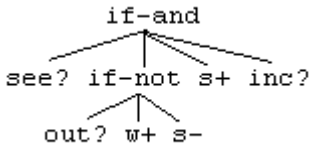


Figure 2. An example of a robot program

It is shown in Chongstitvatana and Polvichai (1996) that

- the robot system can learn to solve the problem in various settings satisfactorily.
- There are improvement of the fitness in the population, in other words, the learning took place during genetic programming process.
- There are improvement in the quality of solutions (the path which the arm traversed was shorter).

GP has becoming increasingly popular in recent year as a method for solving complex search problems in a large number of disciplines : classify protein segment (Koza and Andre, 1996), image processing (Daida et al, 1996), cellular automata (Andre, 1996), electronic commerce (Hinchleffe et al, 1996), information filtering (Zhang et al, 1996) and

chemistry (Handley, 1996). The applications of GA and GP to robot problems are numerous (Khoogar et al, 1989; Koza and Rice, 1992; Chan and Zalzala, 1993; Cliffe et al, 1993; Floreano and Mondada, 1994; Thompson, 1995; Koza et al, 1996). Although most of the work in GA and GP use the simulated world to perform learning, the problem of transferring the result from the simulated world to the real world has been widely recognised (Brooks, 1994; Doringo, 1995; Ito et al 1995). This is called the problem of robustness of the solutions. Chongstitvatana and Polvichai (1996) reported that the solution is satisfactory but not robust. The initial condition for the actual run of a robot program in the real world must be exactly the same as in the simulation, even a small deviation can lead to failure to reach the target.

3.3 Robust solutions

The solutions that are generated by genetic programming method oftenly will fail when there are changes in the operational environment (such as robots working in the real world) even when the change is small. An example can be drawn from our previous work (Chongstitvatana and Polvichai, 1996), in a visual reaching task, genetic programming is used to generate robot programs that control an arm to reach a target avoiding the obstacles, if there are small changes such as an obstacle is moved from its position, or the control of a real robot miss the step (due to random noise), the solution generated from genetic programming will fail to work. This is because genetic programming procedure relies on a simulated world to evaluate and to "evolve" the solution. The accuracy of the world model is an important factor for the success.

We characterise the uncertainty (changes in the operational environment) that occurs in the domain of robotics as following :

- uncertainty in control : when issuing a command to the end-effector, sometime it fails to act properly.
- uncertainty in sensing : the data acquired from sensors are almost always noisy and therefore has a high degree of uncertainty. Measuring distance of an object gives fluctuated reading even when the object is stationary.
- uncertainty in world model : the reconstruction of the model of the world inside the robot and the theory about the action and change are hardly complete. Theory can not predict the consequence of some action reliably. The world can also change, such as some object moved from its previous position.
- uncertainty from interaction : the robot can interact with other object. Pushing a box will change its position, bumping into other mobile robot will definitely change “other” robot behaviour. The problem also arises when there are more than one active robot. An example, for a problem of generating programs for one mobile robot to search for a target, the solution will be different for one robot and many robots. In the latter case, a mobile robot must avoided collision with other mobile robots.

The solution which can tolerate these uncertainties is said to be robust.

4. Research methods and scopes

We attack the problem of how to use genetic programming to generate robust solutions for robot learning from two perspectives : on-line learning and introducing perturbation into simulation. The robot learning problems that we study are

- learn visual reaching task
- learning finite state automata
- learn to evolve robot programs in the real world

These are the problems that have been extensively studied and have some well known solutions (Cliff et al, 1993; Floreano and Mondada, 1994; O'Sullivan et al, 1995; Brooks' COG project at MIT, 1996). Our work is different in term of solving these problems with different constraints. The previous works mostly rely on accurate world model, disallow uncertainty. Our work learns the world model and allow uncertainty in the system, in fact our objective is for the system to be able to tolerate the uncertainties.

4.1 On-line learning

To cope with uncertainty many researchers suggest the use of physical robots to learn in the actual environment of the tasks (Dorigo, 1995; Floreano and Mondada, 1995). The robot will learn by trial and error. This approach is suitable for many learning tasks such as learning the association between sensing-effectors. However, none of the previous work attempt to use genetic programming as the learning method for this approach because it is very likely that it will take too long. The speed limit of a physical robot prohibits the use of learning by trial and error. Chongstitvatana and Polvichai (1996) showed that for a visual-reaching task, it will take 2,000 hours with their equipments to learn the task. We propose to use "memoization", i.e. memorise partial trial and error which has been learned, to speed up the learning processs. The problem of this technique is that it will require the interpolation of the current state (of the system) to a nearest known "learned" state. This interpolation is not linear. We propose the use of searching for an intermediate state combined with a forward prediction to move the present state of the system closer to a known "learned" state. This will be a recursive method. In summary, the "heart" of on-line learning is the use of physical robots to learn in the actual environment of the tasks without excessive number of trial and error. We propose to achieve this by memorise the past experience.

4.2 Introducing perturbation into simulation

Another approach to cope with uncertainty is to subject the "evolved" system to perturbation expecting that the resulting solution will be more tolerant. There are both report of success and failure using this approach. Ito et al (1995) reported success, they arranged the simulation so that at the end of every generation they introduced some perturbation (such as moving the obstacles slightly). However, they limit their claim to one kind of problem, i.e. a mobile robot searching for a target. They also claim that their choice of function set is important for the success (we will discuss this later). Reynolds (1994) reported failure, the author tried to improve the robustness by introducing noise into simulation at every step. The result was that there was no solution found for this corridor-following problem (a mobile robot moving in a connected corridors environment). The author suggested that a

larger number of population may be required or to use a simpler system (reduce the number of function and terminal). Experience from our previous work, we notice that we can introduce limited perturbation "in between" generation and still find solutions (i.e. during a generation, we kept everything constant). Another observation was that we can take a successful individual and continue to "evolve" it for a new environment which is changed only slightly from the previous one. This way we can find the new solution faster than starting from initial random population. We conjecture that solutions from genetic programming can "memorise" the past experience. Therefore, by controlling the amount of perturbation to be introduced into the simulation, we can achieve a more robust solution.

These two approach seem to be in the opposite extreme. There is a possibility to create a "hybrid" system in which use the advantage from both approaches. One can "evolve" solutions quickly under simulation (because there is no severe speed limit of physical devices) then use those solutions as initial population to do on-line learning. This is totally new area which we find no other work to refer to presently.

Another interesting observation is that in applying genetic programming method to a particular problem, the choice of function and terminal set is an important factor for success. For example, in robotics the terminal set will be primitive commands for the robot to act and sense, therefore the terminal set will be restricted by physical robots. The only variable left is the choice of function set, it should be an important factor for success. As we mentioned earlier, Ito et al (1995) claimed that his success depended on the choice of the function set. We will investigate his claim in a more general domain, i.e. our learning task will be more general than the one he reported. In our previous work (Chongstitvatana and Polvichai, 1996; Polvichai, 1996), the function set is {IF-AND, IF-OR, IF-NOT} which is most basic but complete. There are many other possibilities, even the use of index memory as a function (Teller, 1994) has been reported.

5. Impact of the research to Thailand development

The impact of this project will be mostly the establishment of research activities in the area of evolution computing in Thailand. The genetic programming technique itself has a very wide range of applications. The most attractive application is in the field of image processing (including pattern recognition such as OCR). In robotics, an appropriate application of robot learning concept can yield a higher performance in automation which might prove to be very cost effective for many applications in industries.

References

1. Andre, D., Bennett, F. and Koza, J. “Discovery of genetic programming of a cellular automata rule that is better than any known rule for the majority classification problem”. In Proc. of 1st Annual Conf. Genetic Programming, Stanford, Cambridge, MA., MIT press, pp.3-11, 1996.
2. Barto, A., Sutton, R., Anderson, C. “Neuronlike adaptive elements that can solve difficult learning control problems”. IEEE trans. on SMC, 13(5):834-846, 1983.
3. Brooks, R., "Artificial life to actual robots", in Proc. of 1th European Conf. on Artificial life, MIT press,1991, pp3-10.
4. Chan, K., Zalzala, A.,"Genetic-based minimum-time trajectory with torque constraints", IEE Colloquium on genetic algorithms for control systems engineering, digest no. 1993/130, 1993, pp.4/1-3.

5. Chongstitvatana, P. “The design and implementation of vision-based behavioural modules for a robotic assembly system”, PhD thesis, the department of artificial intelligence, Edinburgh university, 1992.
6. Chongstitvatana, P., “Vision-based behavioural modules for robotic assembly system”, Proc. of 6th IEEE Inter. Conf. on Tools with Artificial Intelligence, New Orleans, 1994.
7. Chongstitvatana, P. and Polvichai, J. "Learning a visual task by genetic programming", in Proc. of IEEE/RSJ Int. Conf. on Intelligent robots and systems (IROS96), Osaka, 1996.
8. Cliff, D., Harvey, I. and Husband, P. “Exploration in evolutionary robotics”, Adaptive behavior 2(1):73-110, 1993.
9. Connell, J. and Mahadevan, S. Robot learning, Kluwer academic pub., 1993.
10. Daida, J., Hommes, D., Bersano-Begey, T., Ross, S, Vesecky, J. “Algorithm discovery using the genetic programming paradigm : extracting low-contrast curvilinear features from SAR images of arctic ice”. In Angeline, P. and Kinnear, K. (eds) Advances in genetic programming vol. 2, MIT press, 1996.
11. Davidor, Y. Genetic algorithms and robotics, World scientific, 1991.
12. Dejong, G., Mooney, R. “Explanation-based learning: an alternative view”, Machine learning, 1(2):145-176, 1986.
13. Dejong, G. “A case study of explanation-based control”. in Proc. of the 12th Int. Conf. on Machine learning, pp.167-175, Morgan Kaufmann, 1995.
14. Dorigo, M. “ALECSYS and the autonomouse: learning to control a real robot by distributed classifier system”, Machine learning, 19(3), 1995.
15. Dorigo, M. “Introduction to the special issue on learning autonomous robots”. In Dorigo, M. (ed), IEEE trans. on SMC, 26:361-364, 1996.
16. Dworman, G., Kimbrough, S. and Cheatham, J. “Bargaining by artificial agents in two coalition games: a study in genetic programming for electronic commerce”. In Proc. of 1st Annual Conf. Genetic Programming, Stanford, Cambridge, MA., MIT press, 1996.
17. Floreano, D., Mondada, F. “Automatic creation of an autonomous agent : genetic evolution of a neural-network driven robot. In Cliff, D. et al (eds). From animal to animat 3, Proc. of 3rd Int. Conf. on simulation of adaptive behavior, MIT press, 1994.
18. Franklin, J, Mitchell, T. and Thrun, S. (eds) Robot learning. No 2-3. Kluwer academic press, 1996. Special issue of Machine learning journal, vol.23.
19. Goldberg, D., Genetic Algorithms, Addison-Wesley, 1989.
20. Grefenstette, J., Schultz, A. “An evolutionary approach to learning in robotics”, In Proc. of the workshop on robot learning, 11th Int. Conf. on Machine learning, 1994.
21. Handley, S. “The prediction of the degree of exposure to solvent of amino acid residues via genetic programming”. In Proc. of 1st Annual Conf. Genetic Programming, Stanford, Cambridge, MA., MIT press, 1996.
22. Hinchliffe, M, Hiden, H, McKay, Willis, M., Tham, M. and Barton G. “Modelling chemical process systems using a multi-gene genetic programming algorithm”. In Proc. of 1st Annual Conf. Genetic Programming, Stanford, Cambridge, MA., MIT press, 1996.
23. Holland, J. Adaptation of natural and artificial systems, MIT Press, 1975.
24. Ito, T., Iba, H. and Kimura, M. "Robustness of robot programs generated by Genetic programming", in Genetic Programming 96, MIT press, 1996.
25. Kaelbling, L. Learning in embeded systems, MIT press, 1993.
26. Khoogar, A., Parker, J. and Goldberg, D. "Inverse kinematics of redundant robots using genetic algorithms", in Proc. IEEE Int. Conf. of Robotics and automation, vol.1, 1989, pp.271-276.
27. Koenig, S., Simmons, R. “Passive distance learning for robot navigation”. In Proc. of the 13th Int. Conf. on Machine learning, pp.266-274. Morgan Kaufmann, 1996.
28. Koza, J. Genetic Programming, MIT Press, 1992.
29. Koza, J., Rice, J. “Automatic programming of robots using genetic programming”, in AAAI-92 Proc. of 10th National Conf. on Artificial Intelligence, 1992, pp.194-201.
30. Koza, J., Andre, D. “Classifying protein segments as transmembrane domains using architecture-altering operations in genetic programming”. In Angeline, P. and Kinnear, K. (eds) Advances in genetic programming vol. 2, MIT press, 1996.
31. Koza, J., Bennett, F., Andre, D and Keane, M. “Toward evolution of electronic animals using genetic programming”. Artificial life V : Proc. of 5th Int. workshop on the synthesis and simulation of living systems, Cambridge, MA., MIT press, 1996.
32. Littman, M., Cassandra, T., and Kaelbling, L. “Learning policies for partially-observable environments : scaling up”. in Proc. of the 12th Int. Conf. on Machine learning, pp.362-370. Morgan Kaufmann, 1992.
33. Mahadevan, S., Connell, J. “Automatic programming of behavior-based robots using reinforcement learning”. Artificial Intelligence, 55:311-365, 1992.
34. Mahadevan, S., Proc. of the workshop on robot learning, 11th Int. Conf. on Machine learning, 1994.

35. McClelland, J., Rumelhart, D. (eds) Parallel distributed processing: explorations in the microstructure of cognition, Bradford books, 1986.
36. Mitchell, T., Keller, R., and Kedar-Cabeli, S. “Explanation-based generalization : a unifying view”, Machine learning, 1(1), 1986.
37. Mitchell, T. Machine learning, McGraw Hill, 1997.
38. Morowitz, H. and Singer J. (eds) The mind, the brain and complex adaptive systems, Proc. vol 22 Santa Fe Institute, Studies in the sciences of complexity, Addison-Wesley, 1995.
39. Nehzmov, U., Mitchell, T. A prospective student's introduction to the robot learning problem. Tech. Report UMCS-95-12-6, U. of Manchester, 1990.
40. O’Sullivan, J. and Mitchell, T. and Thrun, S. “Explanation based learning from mobile robot perception”. In Ikeuchi, K. and Veloso, M. (eds) Symbolic visual learning, Oxford university press, 1995.
41. Polvichai, J., Robot learning by genetic programming, M.Eng. thesis, the department of computer engineering, Chulalongkorn university, Thailand, 1996. (in Thai).
42. Pomerleau, D. “Neural network based autonomous navigation”, in Thrope, C. (ed) Vision and navigation: the CMU Navlab, Kluwer academic pub, 1990.
43. Puterman, Markov decision processes : discrete dynamic stochastic programming, John Wiley, 1994.
44. Quinlan, J. Induction of decision trees. Machine learning, 1(1):81-106, 1986.
45. Reynolds, C., "Evolution of obstacles avoidance behavior : using noise to promote robust solutions", in Kinnear, K., (ed), Advances in Genetic Programming, MIT press, 1994.
46. Russell, S., Parr, R. “Approximating optimal policies for partially observable stochastic domains”. In Proc. of the 14th IJCAI, 1995.
47. Sutton, R. “Integrated architecture for learning, planning, and reacting based on approximating dynamic programming”. in Proc. of the 7th Int. Conf. on Machine learning, pp.216-224. Morgan Kaufmann, 1990.
48. Teller, A. “Genetic programming, indexed memory, the halting problem, and other curiosities”. In Proc. of the 7th Annual Florida AI Symposium, IEEE press, 1994, pp.270-274.
49. Thompson, A. “Evolving electronic robot controllers that exploit hardware resources”. In Moran, F. et al. (eds). Advances in artificial life, 3rd European conf. on artificial life, pp.640-656, 1995.
50. Watkins, C., Learning from delayed rewards, PhD thesis, King's college, Cambridge, England, 1989.
51. Zhang, B., Kwak, J. and Soper, A. “Building software agents for information filtering on the internet: a genetic programming approach”. In Proc. of 1st Annual Conf. Genetic Programming, Stanford, Cambridge, MA., MIT press, 1996.

Progress report year 1998

Using perturbation to improve robustness of solutions generated by genetic programming for robot learning

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

- Propose a method to improve robustness of the robot programs generated by genetic programming.
- Main idea is to perturb the simulated environment during evolution of the solutions.
- Test this idea using the problem of navigating a mobile robot in an unknown cluttered environment.

Definition

Robustness continue to perform well under change in environment.

Fragility

- Simulation vs Real-world
- Initial condition
- Accuracy of the simulation model

Example

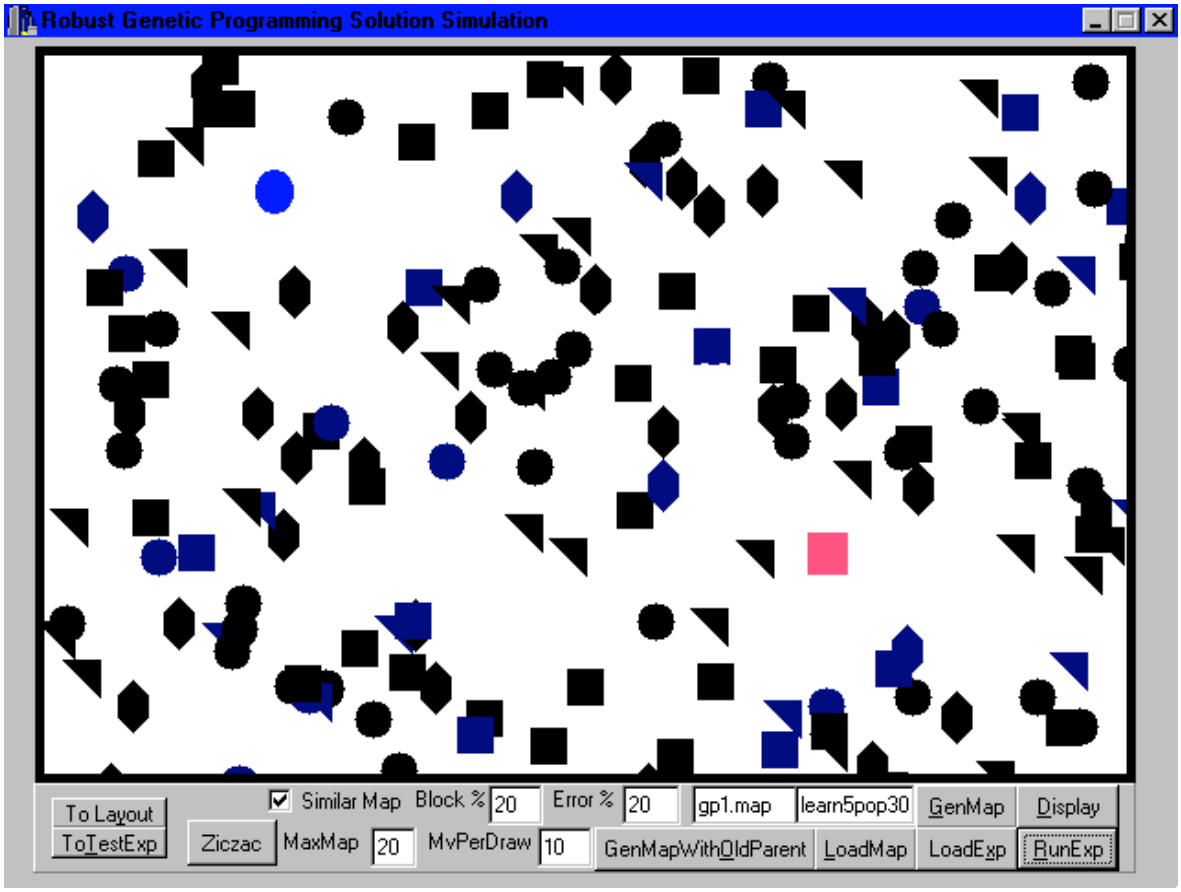
A visual reaching task, GP is used to generate robot programs that control an arm to reach a target.

1. Approach to improve robustness

- Learn in the actual environment (take a lot of time)
- Make simulation more accurate (sample the real-world)
- Use perturbation in the simulation

  The resulting robot programs are more robust because they have been evolved to tolerate the changes in their environment.

2. Experiments



The terminal set is { `move, left, right, isnearer?` }.

The function set is { `if-and, if-or, if-not` }

The fitness measure

$$f = kd + m$$

d = Euclidean distance

k = 10,000

m = the number of moves

The parameters for GP run

population size : 960

maximum number of generation : 200

Selection : Elitism best 30

Crossover : all possible pairing of the parents 900

Mutation : 30

3. Using Perturbation

Experiment 1

Vary the number of training

E' = E + perturbation

the percent of disturbance (*e*).

$$e = \frac{number\_of\_moved\_obstacles}{total\_number\_of\_obstacles}$$

$$f = \sum f_n$$

e = 20 %

n = 1, 5, 10, 15, 20.

Experiment 2

Vary the magnitude of perturbation during the training.

e = (10, 20, 30, 40, 50).

The number of training environment is 10.

Robustness measurement

- Selecting the best individual from the maximum generation and evaluate it under 1000 new environments that are variant of the original.
- Robustness is the number of success of that individual under these new environments.
- The disturbance during the measurement of robustness is 10, 40 and 100%.

Analysis

- The dynamic execution of the program can be represented by the sequence of the robot commands (c_n). This sequence (s), which will be called "trace", is the result of evaluating the robot program, i.e. running it in the simulation.
- The set of all traces (S) is the collection of robot behaviour.
- S represents the interaction of the robot program to the environment, the response of the robot to the different situation.

$$s = (c_1, c_2, c_3, \ldots) \quad (3)$$

$$S = \{ s_1, s_2, s_3, \ldots \} \quad (4)$$

- L is the set of all S in all training environments. L represents a collection of "learned" behaviours.

$$L = S_1 \cup S_2 \cup \ldots S_n \quad (5)$$

- By training in many environments, the robot program encodes "learned" behaviours and hence is able to act properly in a perturbed environment to achieve its goal.
- Figure 3 shows that $| L |$ is increased with the number of training environment (T).
- We measured the similarity (V) of a trace A to L by

$$V = \frac{|L \cap A|}{|A|}$$

V represents how much the behaviour of a robot program is similar to the "learned" behaviours.

- Figure 4 shows that robustness varies with V.

4. Conclusion

We have showed that robustness can be increased by increasing the number of training environment and by increasing the magnitude of disturbance during training. A solution is robust because it encodes the learned behaviour from the training hence it can act effectively in many environments. The results of the experiments demonstrate clearly the effectiveness of the proposed scheme.

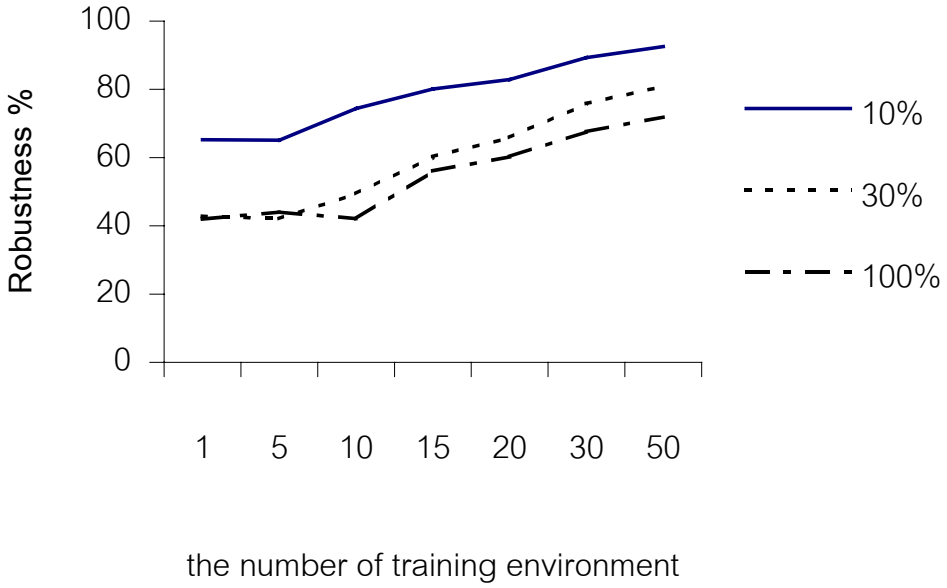


Figure 2 Robustness increases with the number of training environment

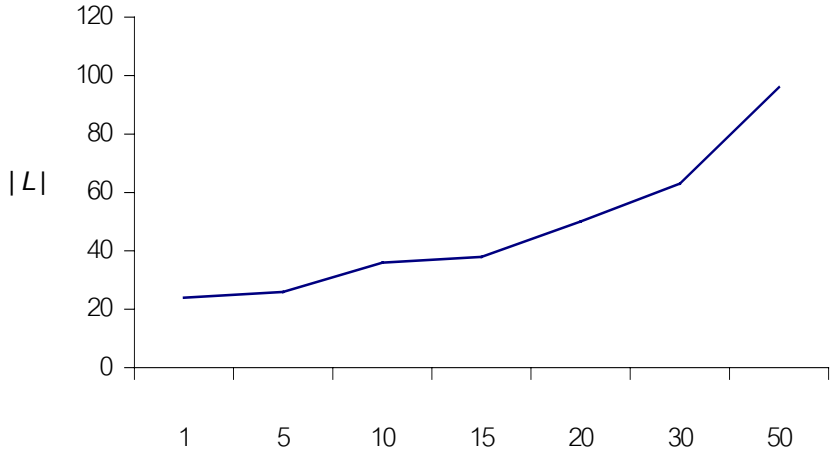


Figure 3 Learned behaviour increased with the number of training environment

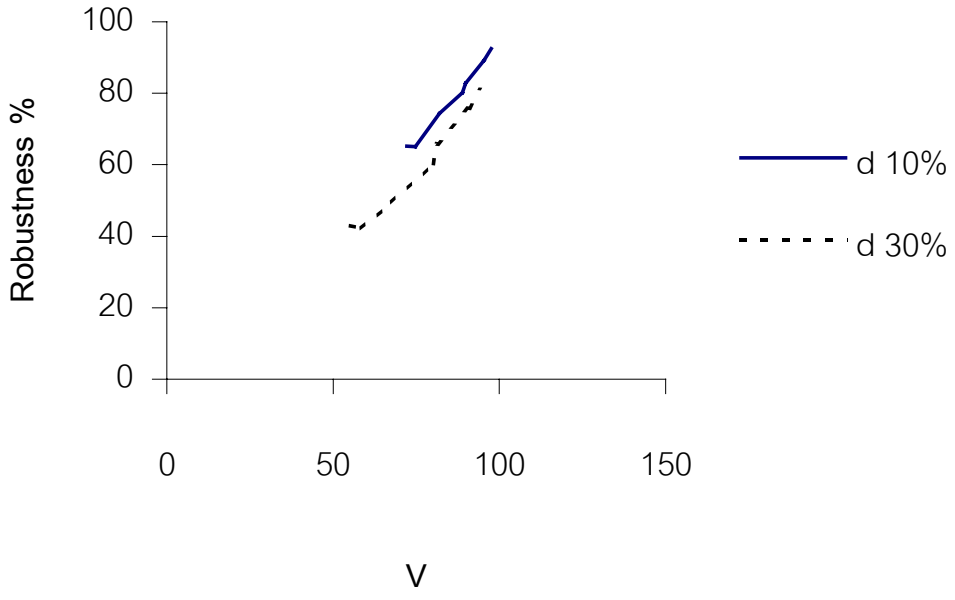


Figure 4 Robustness varies with V

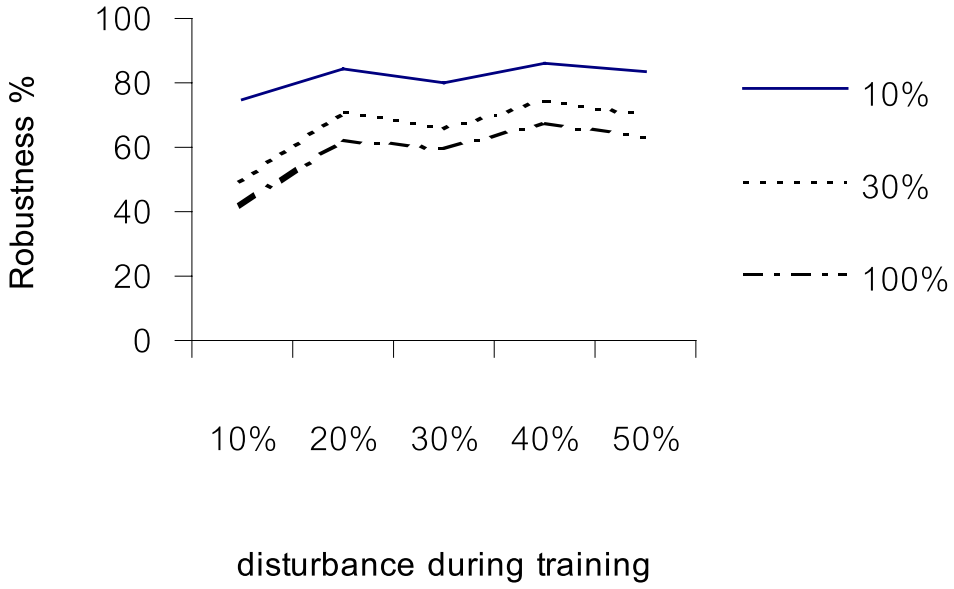


Figure 5 Robustness varies with disturbance during training

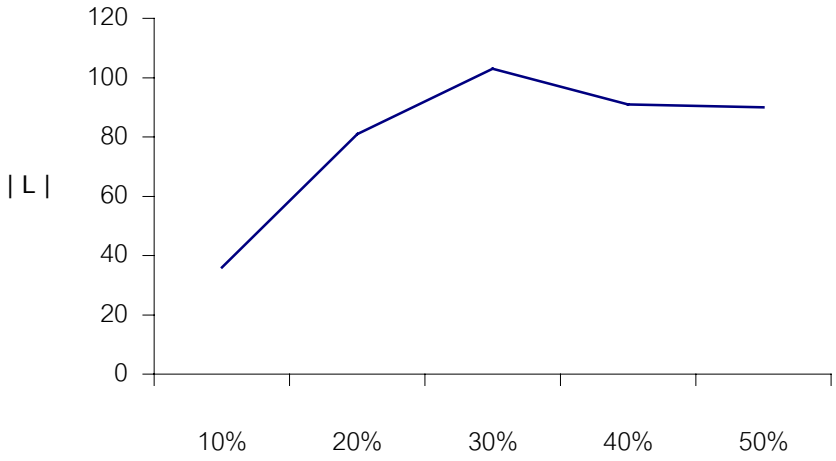


Figure 6 Learned behaviours increased with d_t

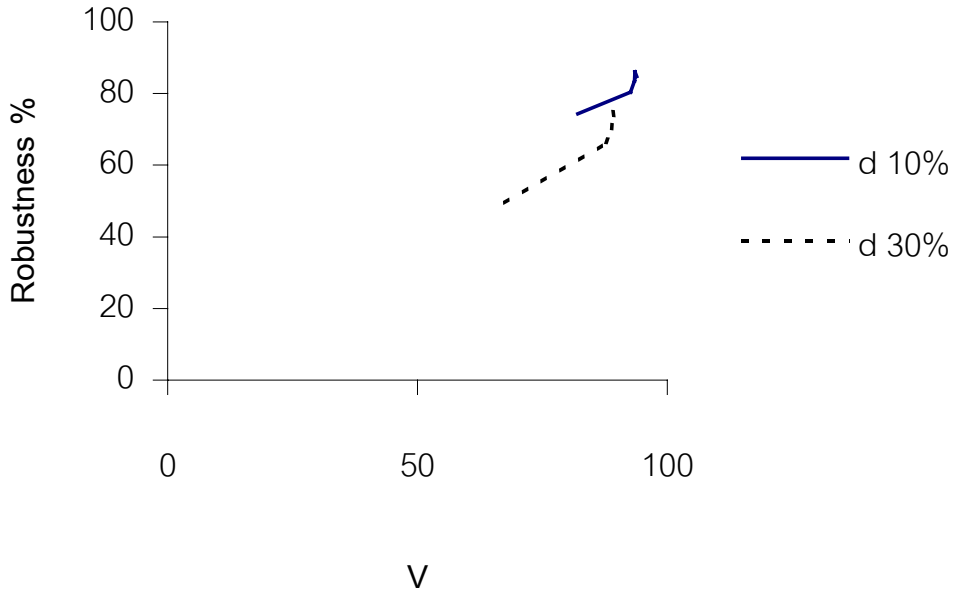


Figure 7 Robustness varies with V

Progress Report year 1999

Learning the environment

Prabhas Chongstitvatana
Department of Computer Engineering
Chulalongkorn University
prabhas@chula.ac.th

Abstract

This year work is concentrated on the on-line learning. We attack the problem of modeling the environment that is, assuming a robot can sense and act in the world, we would like to model the world such that we can predict the consequence of the robot's action. This problem can be abstracted as the *learning automata* problem. We conduct experiments centred on generating Finite State Machine from the observed sensing/action sequences. The result shows that the correctness of the generated circuits depends on the length of the observed sequence. The correctness can be improved using multiple sequences.

1. Introduction

Learning deterministic finite automata can be viewed as a type of learning from the environment. The task of the learner is to construct a model of the environment to predict the consequences of its actions. An environment E can be defined as a Moore machine. A model M of the environment E is defined as (B, Z, S, ϕ , t), where :

- B is a set of basic actions
- Z is a set of percepts, (the robot's sensor reading)
- S is a set of model states
- ϕ is a function from S x B to S, and is the transition function of M
- t is the current model state of M. When a basic action b is applied to M, t is updated to be ϕ(t, b).

We learn M by Genetic Algorithm (GA). We regard learning M as mimicking a sequential circuit by observing its input/output sequence. GA searches for circuits that represent the desired state transition function. The target of our synthesis is a type of Programmable Array Logic which is commercially available as GAL. We are able to synthesize various types of synchronous sequential logic circuit such as counter, serial adder, frequency divider, modulo-5 detector and parity checker.

2 Related work

The conventional method to synthesize a sequential logic circuit requires knowledge of the circuit's behaviour in the form of a state diagram. A sequential network is a common starting point for sequential synthesis system such as Berkeley Synthesis System SIS [1]. In our case, circuit specifications are in the form of partial input/output sequences which are not

suitable to synthesize a circuit by the conventional method. Genetic Algorithm (GA) [2,3] is used to search for circuits that represent the desired state transition function. Additional combination circuits that map states to the corresponding outputs are synthesized by conventional methods. The simulated evolution has been used to synthesis finite state machine (FSM) in [4,5] where the resulting FSM can predict the output symbol based on the sequence of input symbols observed. In contrast to representing circuits as FSMs [6] proposes the automated hardware design at the Hardware Description Language level using GA. [7] describes the evolution of hardware at function-level based on reconfigurable logic devices. [8,9] evolved circuits at the lowest level, in the actual logic devices, using real-time input/output. Our work is similar to [2,3] in the use of FSM but we use FSM as the *model* of the desired circuit behaviour. We aim to evolve the circuit at the logic device level similar to [9]. The following sections describe our synthesis method, the experiment in synthesizing various simple sequential circuits and the analysis of the result.

3 Synthesizer

We use Programmable Logic Device (PLD) as our target structure. For simplicity, we used GAL structure which is composed of rows of two-level sum-of-product Programmable Array Logic (PAL) connected to D flip-flops. The implemented model has four logic terms per one flip-flop, and has a total of four flip-flops as shown in Fig. 1. A circuit is specified by a linear bit-string representing all connection points, which is entirely 256 bits in length.

The outline of the synthesizer's steps is as follows:

1 sample a partial input/output sequence from the target circuit
2 use the sequence as inputs of the synthesizer program
3 verify the resulting circuit if the run yields a solution within 50,000 generations (because GA is a probabilistic algorithm, not all runs are successful in yielding a solution by the specific generation)

The above steps are repeated 50 times for each sampling instance. In this experiment, we used at least 3 different sampling instances for each problem. All of the results are collected and will be used in the analysis.

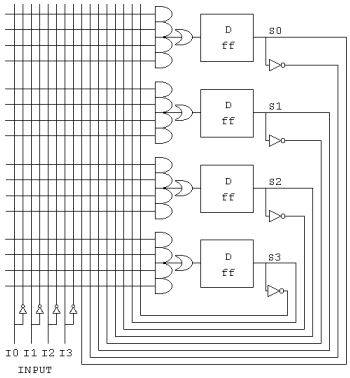


Fig. 1. GAL structure used in the experiment

3.1 Genetic Operations

Each individual is represented by a 256-bit bit-string. We defined genetic operators as follows:

1 Reproduction: Ten new offsprings survive in the next generation by selecting the first 10 fittest individuals ordered by combined rank method [10] (calculated from each individual's fitness rank and its diversity rank).
2 Crossover: More individuals being added to the population are produced by uniform crossover [11]. All possible pairs among 10 already selected individuals are used to produce new offsprings. That is, we will have 90 new individuals.
3 Mutation: Last 10 individuals being added are mutated version of the first 10 selected individuals. The mutation process is controlled so that it changes exactly 5 bits of each individual.

By using the described operators, we can compare our evolving process to Simple Genetic Algorithm (SGA) [3] with these parameters.

population size = 110
crossover rate = 90/110 = 0.8182
mutation rate = 5/256 = 0.0195

We also add some more constraints. First, we avoid creating a product term which always be "0". Second, connection points to unused input signals are left unwired. This reduces the search space of the problem and lets the program concentrate on the connection points that do affect the function of the circuit.

3.2 Fitness Evaluation

An individual is evaluated by the following steps:
1 feed one input to the circuit and clock the circuit
2 next state of the circuit would be mapped with the corresponding output, record the number of times the state has been mapped to both output "0" and "1"
3 repeat steps 1 and 2 until the end of the sequence

After the sequence is completed, the fitness value of the individual, F, is computed by:

$$F = \sum_{i=0}^{i=S-1} f_i \qquad \text{where} \quad f_i = \begin{cases} \max(p_i, q_i) & ;\ p_i = 0 \text{ or } q_i = 0 \\ -\min(p_i, q_i) & ;\ \text{otherwise} \end{cases} . \qquad \textbf{(1)}$$

where
f_i is the fitness value of state i
p_i is the number of times in which state i has to be mapped with output "0"
q_i is the number of times in which state i has to be mapped with output "1"
S is the number of states, equal to 16 for the GAL structure

3.3 Size of Input/Output Sequences

A partial input/output sequence, which is used as a circuit's specification, is attained by generating a sequence of inputs, feeding each one to the target machine and then recording the corresponding output that the machine gives. To be a general approach and yield a simple analysis, the input sequence is created at random with uniform distribution (i.e. at any time, the probability that the input bit be "0" is as high as of "1").

The input sequence should be long enough to exercise all aspects of the circuit's function. In other words, it should be able to test all paths of the state diagram of the circuit. As mentioned earlier in this paper, we use the GAL structure which can be programmed as a 16-state state machine, larger than the desired circuit's need. Therefore, the input sequence should also be long enough to exercise all paths of any 16-state circuit. If the sequence is too short, it may cause an ambiguity in describing the desired circuit. In this paper, we will call

the length of the input sequence which is long enough to describe only the desired circuit as *lowerbound length* and call the length of the input sequence which is long enough to exercise any 16-state circuit as *upperbound length*. We can find the proper size of the input sequence by making an analogy to a dice rolling problem. Consider rolling a dice, how many times do we have to roll it, until all of its faces come out? This problem is called *waiting times in sampling* [12], in which we can find the *expected value* of the number of times by the following formula.

$$E(k,n) = \frac{n}{n-k+1} + \frac{n}{n-k+2} + \cdots + \frac{n}{n} \qquad (2)$$

$$E(n,n) = n\left\{\frac{1}{1} + \frac{1}{2} + \cdots + \frac{1}{n}\right\} . \qquad (3)$$

Consider a circuit that has i bit inputs, giving possible $I = 2^i$ patterns, and has S states. We assume that at any time, the probability that the circuit be in any state is equal. So, from the starting state, we expect the number of state transitions to be $E(S,S)$ to traverse through all states of the state diagram. And at any state, we expect the number of input patterns to be $E(I,I)$ to traverse through all paths from that state. Therefore, we would expect the number of input patterns to be $E(S,S) \times E(I,I)$ to traverse through all paths of the state diagram starting from any state.

Table 1. Description of circuits in the experiment

Circuit	# of input bits	# of states		lower bound length		upper bound length
		Moore	Mealy	Moore	Mealy	
Frequency Divider	0	8	8	22	22	55
Odd Parity Detector	1	2	2	9	9	163
Modulo-5 Detector	1	6	5	45	35	163
Serial Adder	2	4	2	70	25	451

The experiment was done with many input/output sequences of different lengths for each problem. The selected lengths are 10, 100, 1000, lowerbound and upperbound. For each length, at least 3 different random sequences of input/output are used. And the synthesizer was repeatedly run 50 times on each sequence. Thus, we had at least 3×50, equal to 150, independent runs for each length of the sequence for one problem. Table 1. shows the details of desired circuits and the calculated lengths of each problem.

4 Experiment and the Results

Fig. 2 shows the evolution of a serial adder. The circuit that performs accordingly to the input/output sequence appears in the 53rd generation. We know it is a correct serial adder after we verified it. We could observe many redundant states in the resulting circuit because the given space (16-state machine) is larger than the solution.

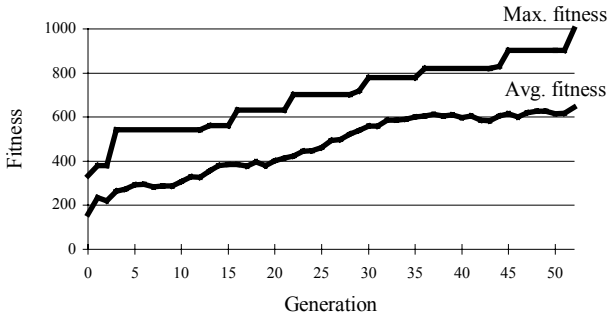


Fig. 2. Evolution of a Serial Adder (Mealy's model, size of input sequence = 1000)

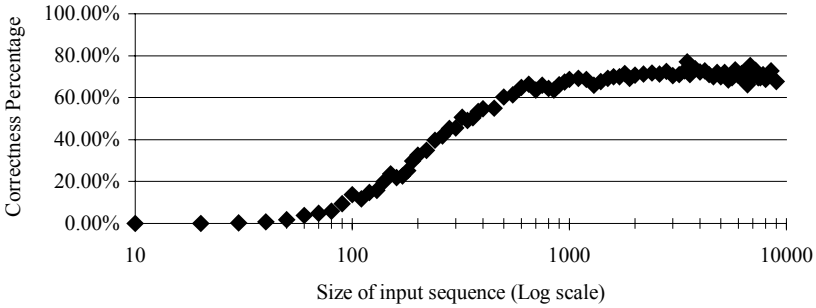


Fig. 3. Correctness and size of input sequence (Serial Adder, Mealy's model)

We define the correctness percentage as

$$\text{Correctness Percentage} = \frac{\text{number of runs yielding correct solutions}}{\text{number of runs yielding solutions}} . \qquad \textbf{(4)}$$

Fig. 3 shows the relation of the size of input sequence to the percentage of correct results of synthesis from the sequence of that size. We used five random input sequences for each point and run each input sequence for 100 times to make the average correctness. The longer input sequence increases the correctness. However, the correctness percentage becomes saturated at the large size of input sequence.

Table 2. The summary of computational effort

Circuit	Effort	
	Moore	Mealy
Frequency Divider	770	440
Odd Parity Detector	1,210	1,760
Modulo-5 Detector	87,967,440	7,018,000
Serial Adder	3,035,120	26,730

5 Conclusion

This report described synthesis of synchronous sequential logic circuit from a partial input/output sequence. GA was applied to synthesize a circuit that based on Moore and Mealy's model on the GAL structure. We can approximate the suitable size of the input

sequence to yield high correctness and low effort. This work can be extended in many ways. One major aspect is the enhancement of the evolutionary process in both effort and time. We are working on an implementation of this work in real hardware to realize an on-line evolware [14,15].

References

1. Sentovich, E., Singh, K., Moon, C., Savoj, H., Brayton, R. and Sangiovanni-Vincentelli, A.: Sequential circuit design using synthesis and optimization. Proc. of Int. Conf. on Computer Design (1992)
2. Holland, J.: Adaptation in natural and artificial systems. MIT Press (1992)
3. Goldberg, D.: Genetic Algorithm in search, optimization and machine learning. Addison-Wesley (1989)
4. Fogel, L.: Autonomous Automata. Industrial Research **4** (1962) 14-19
5. Angeline, P., Fogel, D., Fogel, L. : A comparison of self-adaptation methods for finite state machine in dynamic environment. Evolution Programming **V**. L. Fogel, P. Angeline, T. Back (eds). MIT Press (1996) 441-449
6. Mizoguchi, J., Hemmi, H., Shimohara K.: Production Genetic Algorithms for automated hardware design through an evolutionary process. Proc. of the first IEEE Int. Conf. on Evolutionary Computation (1994) 661-664
7. Higuchi, T., Murakawa, M., Iwata, M., Kajitani, I., Liu, E., Salami, M.: Evolvable hardware at function level. Proc. of IEEE Int. Conf. on Evolutionary Computation. (1997) 187-192
8. Thompson A., Harvey, I., Husbands, P.: The natural way to evolve hardware. Proc. of IEEE Int. Conf. on Evolutionary Computation (1996) 35-40
9. Thompson, A.: An evolved circuit, intrinsic in silicon, entwined with physics. Proc. of the First Int. Conf. on Evolvable Systems: From Biology to Hardware (ICES96). Lecture Notes in Computer Sciences. Springer-Verlag (1997)
10. Winston, P.: Artificial Intelligence. Addison-Wesley (1992) 505-528
11. Syswerda, G.: Uniform crossover in Genetic Algorithms. Proc. of the Third Int. Conf. on Genetic Algorithms, J. D. Schaffer (ed). Morgan Kauffman (1989) 2-9
12. Feller, W.: An introduction to probability theory and its applications vol. **I**. Wiley (1968) 224-225
13. Koza, J.: Genetic Programming. MIT Press (1992)
14. Tomassini, M.: Evolutionary algorithms. Towards Evolvable Hardware. E. Sanchez and M. Tomassini (eds). vol. **1062** of Lecture Notes in Computer Science, Springer-Verlag, Heidelberg. (1996) 19-47
15. Sipper, M., Goeke, M., Mange, D., Stauffer, A., Sanchez, E., Tomassini, M.: The Firefly machine: Online evolware. Proc. of IEEE Int. Conf. on Evolutionary Computation. (1997) 181-186

Progress Report year 2000

Evolving robot programs in the real world

Prabhas Chongstitvatana
Department of Computer Engineering
Chulalongkorn University
prabhas@chula.ac.th

Abstract

This work presents an automatic method to synthesis robot control programs. The proposed method is based on Evolutionary Computation. The problem of biped robot walking is chosen to test the proposed method. Walking motion is divided into six stages and the evolution of control programs is carried out stage-by-stage. The locomotion is restricted to forward walking on the flat and smooth surface with static balance. The synthesis process consists of both simulation and the experiment with a real robot. The result of the experiment shows that the biped walking is achievable and stable.

Keywords: *Evolutionary computation, biped robot walking, static balance walking.*

1. INTRODUCTION

Automatic programming for a robot to achieve a task has been a long-term goal of robotic research community. Programming a robot by human is difficult and error prone. Modern robots are very complex, some robot has sophisticated mechanisms that enable it to perform human tasks such as the humanoid robot P2 by Honda (Hirai, et al, 1998). The limitation of using these robots is the difficulty in programming them to achieve a desired task.

Evolutionary Computation is a family of algorithms, some of which can produce solutions in the form of "programs". It is applicable to robot problems. Many works have been demonstrated, for example, (Koza and Rice, 1992; Davidor, 1990; Chongstitvatana and Polvichai, 1996). Evolutionary Computation can be regarded as a weak search method. It is effective for a wide range of problems such as symbolic regression, job scheduling, robot control and so on.

Evolutionary Computation is a search method based on population. A number of candidate solutions are evolved generation by generation to converge to a final solution. The search is guided by the measure of goodness of candidate solutions, called "fitness function" which is defined for a particular problem to be solved.

Many problems in robot program synthesis that have been attempted using evolutionary computation are the problems that have low number of degree of freedom and mostly are the work in simulation. This is because of the high cost of computation and the huge number of candidate solutions to be evaluated. It is well known that transferring the solution from simulation to the real world is not very successful (Brooks, 1991a). Many aspects of the real world can not be sufficiently simulated. To improve the success rate, the real world should

be engineered such that the simulation can predict the effect in the real world with high degree of accuracy. This is difficult if not impossible in many tasks which robots are intended to be used.

This work proposes to synthesis programs for a biped static walker. This task is chosen because it contains high degree of freedom. A walking robot is interesting because it can travel in many terrains that are not accessible to a typical wheel-based mobile robot. A biped robot is also more appropriate in the area that is constructed for human, such as in a car, in a tunnel, on an elevator etc. A biped is deemed to be more difficult to control than a multi-legged robot as it has to perform balancing with minimum degree of redundancy. Genetic Algorithm is used to synthesis programs. The walking task is divided into stages and the program is synthesized stage-by-stage. In each stage, the solutions from simulation are validated using the experiment in the real world. These validated solutions become the initial state of the next stage of the synthesis. This is the key to improve the transferring of solutions form simulation to the real world. The subsequent sections explain the proposed method in more details.

2. RELATED WORKS

There are many works on generating robot programs. Genetic Algorithms (GA) and Genetic Programming (GP) two of the most popular methods in Evolutionary Computation are widely used. Hirai, et al, (1998) developed a humanoid robot that has full body, head arms, and legs. It could walk perfectly like human, it could walk up and down the staircase, turn left and right, and walk on any surface. This robot, however, used manual programming. Chongstitvatana and Polvichai (1996) demonstrated the automatic generating of robot programs by using Genetic Programming (GP). The robot is 3-joint arm moving in two dimensions. Experiments were performed in simulation, and the results were validated in a real robot.

There are many works on generating robot program to control biped locomotion. Zheng, et al, (1988) developed biped walking from a level surface to sloping surfaces with positive gradients. Inaba, et al, (1995) constructed an ape-like biped that can walk with static balance. Kun and Miller (1997) applied neural network to perform adaptive static balance of biped walking.

Regarding works that apply GA to solve the biped walking problem. Cheng and Lin (1995) developed a walking robot with dynamic balance. In his work, GA is applied to search for control gains and nominal trajectory for a 5-link biped locomotion. The aim is to walk in different constraints, such as, walking on an incline surface, walking at a high speed, and walking with a specified step size. The biped is experimented in simulation. Rodrigues, et al, (1996) used GA to find the minimum torque that is necessary for walking. The fitness function is defined as similarity between the ideal posture and the actual posture. The experiment is also performed in simulation. Arakawa and Fukuda (1996, 1997) focused on using GA to produce a natural motion trajectory and optimize walking energy. The learning system was performed in simulation and the result was confirmed by the real robot.

Most researches experimented only in the simulation. In this work, there are both simulation and real world included in the experiment.

4. EXPERIMENT

The objective is to synthesis the biped robot control program automatically. This work restricts the walking task to the biped that can walk forward on the flat and smooth surface with a static balance.

The experimental biped is 25 centimeter high, and the area of the sole is 4.5×5.0 cm^2. It has two hips, two knees, and two ankles, rotated in sagittal plane (Fig. 3). The biped does not have a torso, but it has a tail moving in frontal plane. The reason for using a tail instead of a torso is that the tail will lower the biped's center of gravity (C.G.), so the biped can keep its balance easier.

An individual contains two fields: a length, and a sequence of walking commands.

Length	a sequence of walking commands

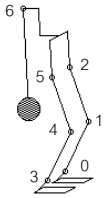


Fig. 3. Biped construction

The sequence of walking commands has the form:

m: r

where 'm' is motor command {0+, 0-, 1+, 1-, …, 6+, 6-}. The biped has 7 motors numbered 0-6. The signs '+' and '-' mean increasing or decreasing angle of the motor by 'r', $0 \leq r \leq 150$.

Walking motion of one step is divided in to six stages (Fig. 4). GA is used to synthesize control program for each stage step-by-step, called "stage evolution" (Brooks, 1991b). With this approach, the fitness function can be set appropriately with the subgoal of each stage. Thus, the final solution can be achieved more rapidly.

The initial biped posture is standing on two feet. In the first stage, the robot shifts its weight to the right foot. In the second stage, it lifts the left leg. The third stage, it lays down the left leg. The fourth stage, the robot shifts its weight to the left foot. The fifth and sixth stage it lifts the right leg and lays it down. After the final stage, the posture is adjusted to be similar to the initial posture. The sequence of control can be repeated to create a continuous walk.

There are two types of fitness function: general fitness function, and particular fitness function. Both fitness functions are minimized function. The general fitness function consists of three variables:

$$Fit = k_1F + k_2R / k_3 T$$

where $F = 1$ when the robot falls otherwise 0, $R = 1$ when the robot turns otherwise 0, T is the duration that the robot can achieve stable walk, k_1, k_2, k_3 are appropriate constants. The general fitness function promotes the behavior that is stable and walk straight without turning.

The particular fitness function for each stage is shown in Table. 1.

$F_1 = \Delta S_R$	$\Delta S_R = \sqrt{(cg_x - pr_x)^2 + (cg_z - pr_z)^2}$
$F_2 = k\Delta z + \Delta y$	$\Delta z = \lvert pl_z - (pr_z + step\_size) \rvert$ $\Delta y = \lvert pl_y - ground \rvert$
$F_3 = \Delta y +$ *penalty*	$\Delta y = \lvert pl_y - ground \rvert$
$F_4 = \Delta S_L$	$\Delta S_L = \sqrt{(cg_x - pl_x)^2 + (cg_z - pl_z)^2}$
$F_5 = k\Delta z + \Delta y$	$\Delta z = \lvert pl_z - pr_z \rvert$ $\Delta y = \lvert pr_y - ground \rvert$
$F_6 = k\Delta y + \Delta z$	$\Delta z = \lvert pl_z - pr_z \rvert$ $\Delta y = \lvert pr_y - ground \rvert$

Table. 1. Particular fitness function for each

where
cg_x, cg_z is the position of C.G. by X and Z axis
pl_x, pl_y, pl_z is the position of center of the left sole by X, Y, and Z axis
pr_x, pr_y, pr_z is the position of center of the right sole by X, Y, and Z axis

step_size is the length of stride (2.5 cm. in the experiment)
penalty is the penalty value if the robot shifts its weight from the right foot.
ground is the position of ground level (Y axis)
k is constant

The motivation for each particular fitness function is as follows. For the first stage, the robot must shifts its C.G. to the right foot. The function F_1 measures the distance between C.G. and the center of the right foot (ΔS_R). In the second stage, the robot lifts the left leg and moves it forward. The function F_2 measures the distance of the left foot in front of the right foot (Δz). The variable Δy controls the height of the left foot from the ground. The *step_size* is used to limit the length of stride to prevent the subsequent difficulty in transferring the weight to the right foot in the fourth stage. In the third stage, the left foot is laid down to the ground. The function F_3 measures the height of the left foot from the ground (Δy). The *penalty* value is used to prevent the robot from shifting its weight to the left foot. If this happens the robot will sway its body to the left side. In the fourth stage, the robot moves its C.G. from the right foot to the left foot. F_4 is similar to F_1 but alternate left and right. The fifth stage is similar to the second stage but alternate left and right. The right foot is not placed forward, the *step_size* is zero. The robot lays down the right foot in the sixth stage. The variable Δz is used to prevent the right leg to move backward or forward.

Evolving the control programs in the simulation is necessary. The experiment with an actual robot takes a very long time as the number of candidates to be evaluated is up to 100,000. However, the solution from simulation alone does not yield programs that works in the real world. The experiment with the real robot is combined in the simulation to increase the success rate. At the end of each stage of evolution, the experiment with the real robot is performed to validate the solutions.

GA is run in each stage of evolution in the simulation (Fig. 5). GA generates the solutions with some variations. As the simulation ignores many aspects of the real world, many solutions from the simulation simply fail. However, some solution has a chance of success. The experiment with the real robot is performed to select only the solutions that work in the real world to be further evolved to the next stage. Each validated result becomes an initial state of the next walking stage. After six stages, the complete solutions will emerge. The number of different solutions from each stage is retained to the next stage hence there are many different complete solutions at the end. This method combines the advantage of simulation (speed) with the advantage of the experiment with the real robot (validity).

For the experiment with the real robot, the human observation is used to score the behavior of the robot. There are 2 types of criteria in observing the real behavior: *general criteria* used in every stages, and *particular criteria* used in each stage. The general criteria judges the stability and the direction of the walk. The observer asks the questions "Does the robot fall?" and "Does the robot turn?". The particular criteria are set differently for each subgoal of each stage, as shown in Table 2.

Stag	particular criteria
1	Is C.G. shifted to the right foot?
2	Is the left leg lift forward?
3	Is the left foot on the ground?
4	Is C.G. shifted to the left foot?
5	Is the right leg move forward?
6	Is the right foot on the ground?

Table 2. Particular criteria for each

The GA parameters are shown in Table 3.

Population size	500
Generation	200
Crossover probability	1.0
Mutation probability	0.001

Table. 3. GA parameters

Because an individual can contain redundant motions, such as moving a joint back and forth or repeating the same joint motion, edit operations are performed after its fitness evaluation. The edit operations are 1) eliminate redundant motions and 2) simplify repeating joint motions. These operations help to maintain the compactness of the representation.

5. RESULT

The robot can walk continuously more than 15 steps, with the speed 40 second per step. Figure 6 shows an example of a full step. It can be seen that the stability of biped locomotion is marginal, especially in the stage 2 - 4.

At the end of each stage in the simulation, 20 individuals are selected to be validated with the real robot. An average number of successful individual in the experiment with the real robot of each stage is 7. We found that even without the general fitness function, the final solution could still be achieved.

The fourth stage is the most difficult stage to evolve. The robot must transfer its weight to another foot. It becomes more difficult when the length of stride is large. The length of stride is determined by the fitness function in the second stage. Sometimes, the unexpected behavior emerges in the fourth stage such as moving the leg backward before the weight transfer.

6. CONCLUSION

In this work, we investigate a method to automatically generate control programs for a walking biped. Walking motion is divided into six stages. GA is used to synthesize the robot control program stage-by-stage. The fitness function is set differently and appropriately in each stage. This work uses simulation combined with the experiment in a real robot. The results show that the real robot can achieve a stable and continuous walk.

The experiment in the real world is used to select and validate the result from the simulation. The cooperation between simulation and real world experiment is the key to achieve a solution that works in the real world.

7. REFERENCES

1. Arakawa T., Fukuda T. (1996). “Natural motion trajectory generation of biped locomotion robot using genetic algorithm through energy optimization”, Proc. of IEEE Int. Conf. on Systems, Man and Cybernetics, Vol. 2, pp. 1495 -1500.
2. Arakawa T., Fukuda T. (1997). “Natural motion generation of biped locomotion robot using hierarchical trajectory generation method consisting of GA, EP layers”, Proc. of IEEE Int. Conf. on Robotics and Automation, Vol. 1, pp. 211 - 216.
3. Brooks R. A. (1991a). “Artificial Life and Real Robots”, Towards a Practice of Autonomous Systems: European Conference on Artificial Life, Paris, France, MIT Press, December, pp. 3 - 10.
4. Brooks R. A. (1991b). “Intelligence without representation”, Artificial Intelligence, 47, pp. 139 - 160.
5. Cheng M.-Y., Lin C.-S. (1995). “Genetic algorithm for control design of biped locomotion”, Proc. of Int. Conf. on Intelligent Systems for the 21st Century, Vol. 2, pp. 1315 -1320.
6. Chongstitvatana, P., Polvichai, J. (1996). “Learning a visual task by Genetic Programming”, Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, Vol. 2, pp 534 - 540.
7. Davidor Y. (1990). “Robot programming with a genetic algorithm”, Proc. of IEEE Int. Conf. on Computer Systems and Software Engineering, pp. 186 - 191.
8. Goldberg D. E. (1989). “Genetic Algorithm in Search, Optimization, and Machine Learning”, Addison Wesley.
9. Hirai K., Hirose M., Haikawa Y., Takenaka T., (1998). “The development of Honda humanoid robot”, Proc. of IEEE Int. Conf. on Robotics and Automation, Vol. 2, pp. 1321 - 1326.
10. Inaba M., Kanehiro F., Kagami S., Inoue H., (1995). “Two-Armed Bipedal Robot that can Walk, Roll Over and Stand up”, Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, Vol. 3, pp. 297 - 302.

11. Koza J. R., Rice J. P. (1992). “Automatic programming of Robots using Genetic Programming”, in Proc. 10th National Conf. on Artificial Intelligence, pp. 194 - 201.
12. Kun A. L., Miller W. T. (1997), “Adaptive Static Balance of a Biped Robot Using Neural Networks", Proc. of the Int. Conf. on Robotics and Manufacturing (IASTED), Cancun, Mexico, May 29-31, pp. 245 - 248.
13. Rodrigues L., Prado M., Tavares P., Da Silva K., Rosa A. (1996), “Simulation and control of biped locomotion-GA optimization”, Proc. of IEEE Int. Conf. on Evolutionary Computation, pp. 390 - 395.
14. Winston, P. (1992). “Artificial intelligence”, Reading MA: Addison-Wesley, pp. 505 - 528.
15. Zheng Y. F., Shen. J., Sias F. (1988). “A motion control scheme for a biped robot to climb sloping surfaces”, Proc. IEEE Int. Conf. on Robotic and Automation, Vol. 2, pp. 814 - 816.

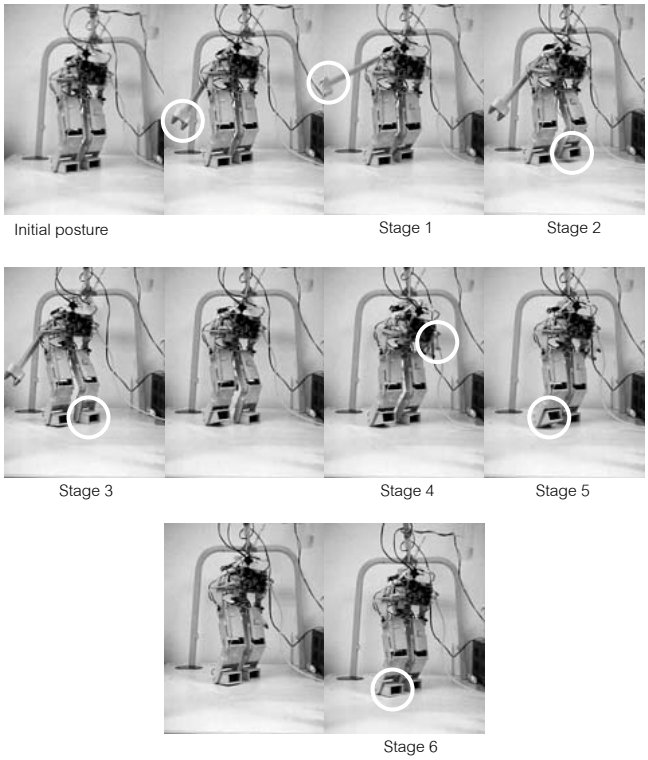


Fig. 6. Result.

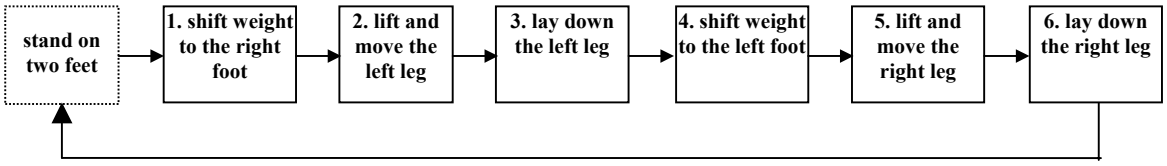


Fig. 4. Six stages of walking motion.

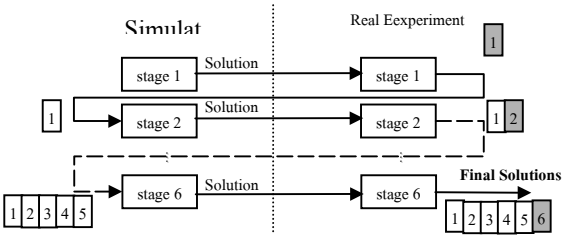
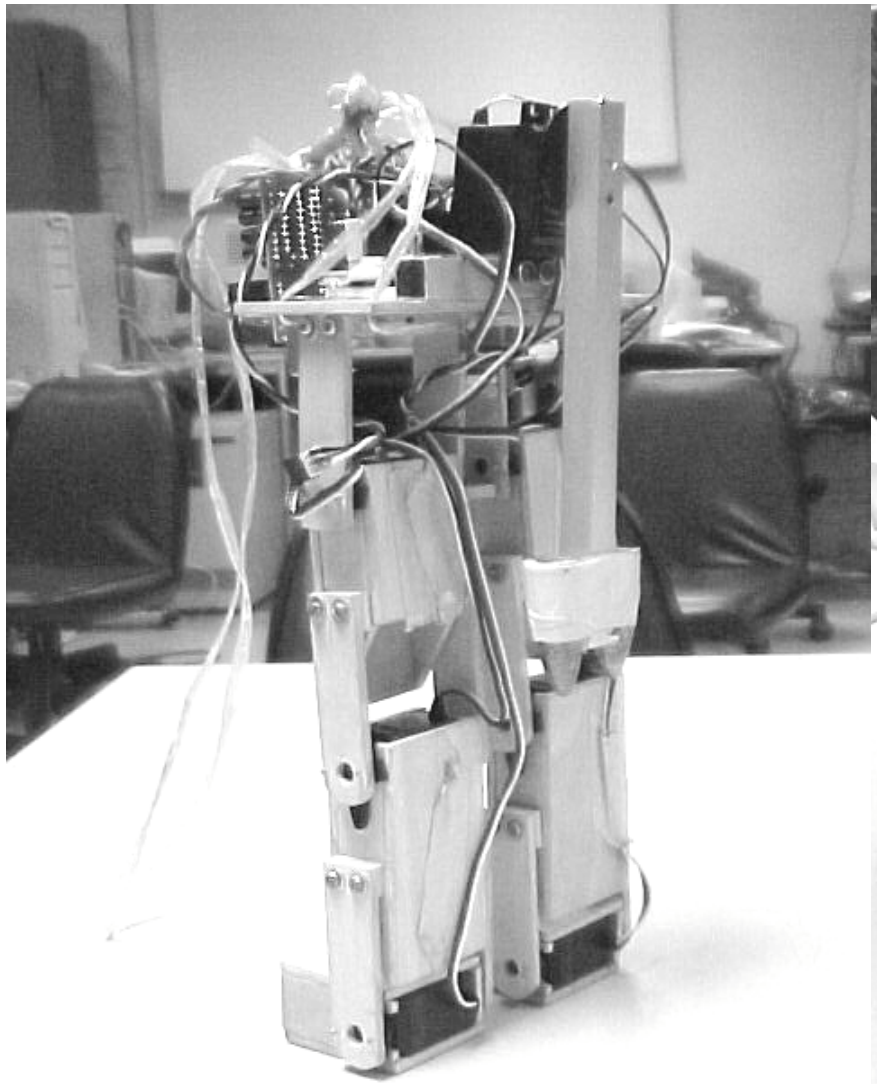
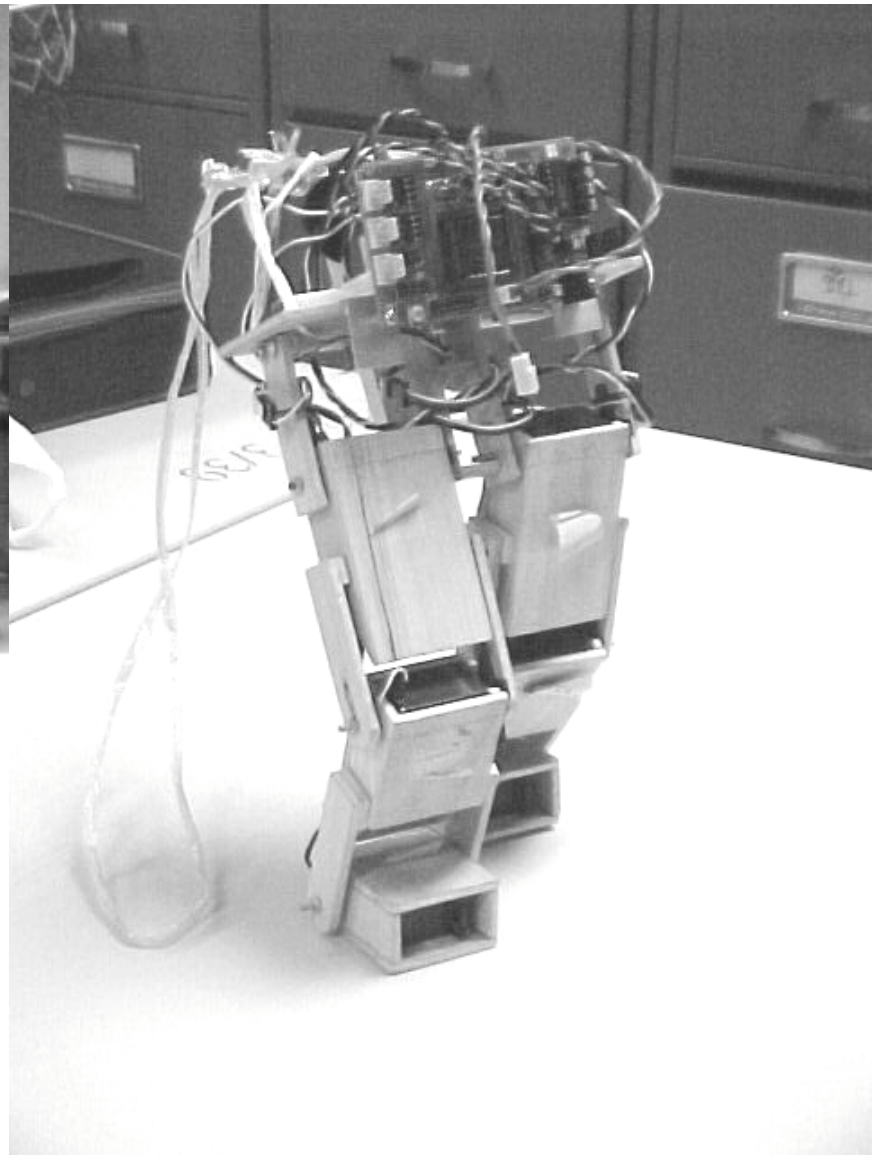


Fig. 5. Simulation + real world





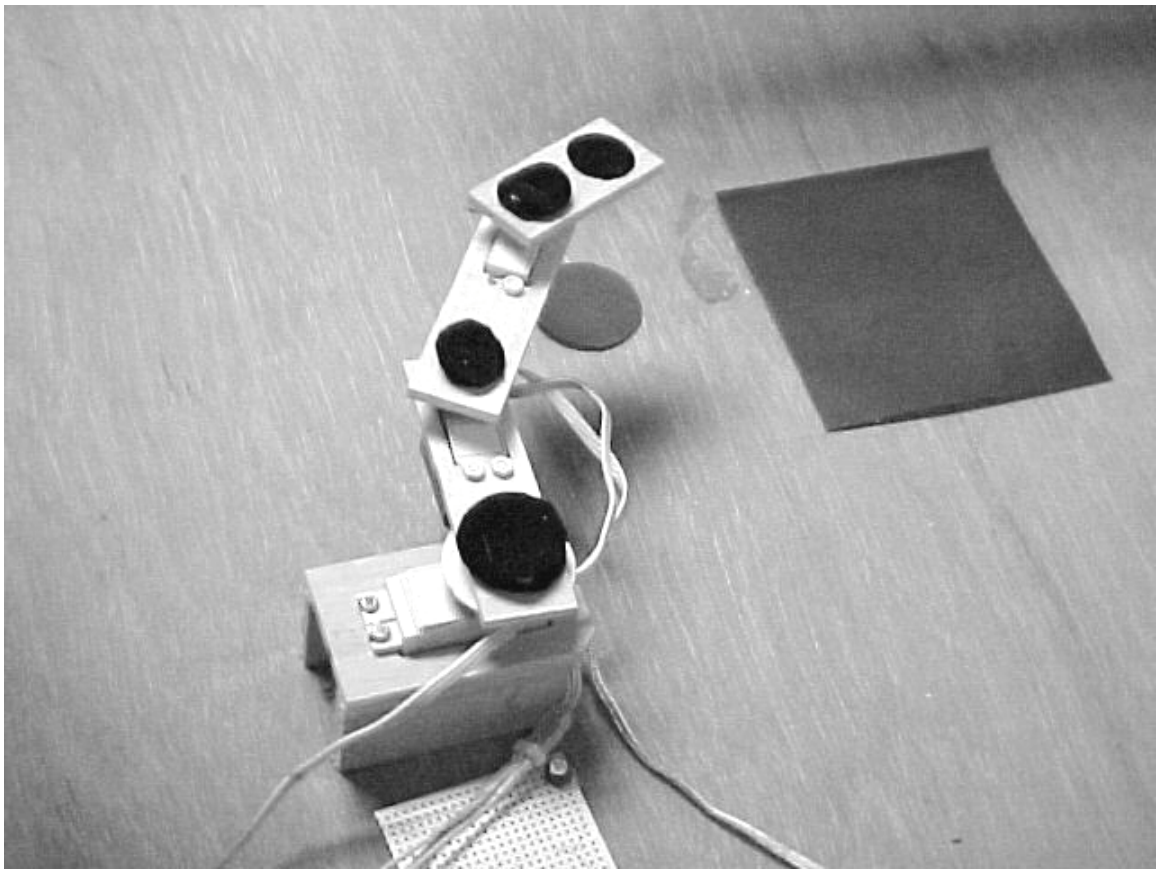


Figure (this page) The experimental setup for visual reaching task, a 3-DOF planar robotic arm. (previous page) The biped robot used in the experiment for evolving robot programs to control a robot to walk. These two experiments are the experiments on evolving robot programs in the real world.