Learning a Visual Task by Genetic Programming

Prabhas Chongstitvatana and Jumpol Polvichai
Department of computer engineering
Chulalongkorn University
Bangkok 10330, Thailand
fengpjs@chulkn.car.chula.ac.th

Abstract

This work describes a hand-eye system that can learn from its experience. The task is to visually guide the hand to reach a target while avoiding obstacles. The motion planning problem is solved by genetic programming. The system learns the forward kinematics by building a look-up table and uses it in the simulation run to generate robot programs that perform the task. The genetically created programs are validated by the actual runs on the robot.

1: Introduction

One aim to achieve general intelligence in the field of artificial intelligence is to have a system that can learn. A learning system can extract information from an unknown environment, especially when an accurate model of environment is not available. Agents having a continuous existence can also learn from experience, they should be able to learn how to solve new problems when the circumstances arise. One learning technique is the genetic programming method. In genetic programming paradigm, genetically breed populations of computer programs are used to solve problems. The individuals in the population are compositions of functions and terminals. An evolutionary process is driven by the measure of fitness of each individual computer program in handling the problem environment.

Our goal is to develop an efficient learning system that is capable of performing the task in the real world. A visually-guide robotic task is a good domain for experimentation in learning. The perceptual task in the world has the characteristic that the relevant laws and interrelationships between a robot and the environment are highly reliable. We adopt the interpretation that a learning system is a system that is capable of making changes to itself over time with the goal of improving its performance on its task. We measure the performance of the system under the real world environment. Our system performs the learning task using the real robot and visual feedback data from the real world.

2: Previous work

Visual feedback has been used to guide robots in hand-eye coordination tasks since the early days of robotics research. Jones [1] demonstrated the use of visual tracking methods to perform block stacking and loose insertion. To perform tasks in a complex world, the motion planning part in a robotic system relies on the geometrical model of the environment and the robot. The system such as HANDEY [2] performs hand-eye coordination tasks in a structured environment in which an accurate model of the world is available. The system plans collision free paths in a configuration space. Because it relies on accurate models of the world the system is unable to cope with uncertainty. The environment has to be engineered to reduce the uncertainty to an acceptable level. The work to extend this paradigm to cope with uncertainty is still an active research area [3].

Another approach is to have the robot system learn the task by itself. In [4], MURPHY, a robot motion planning system is presented. The system approaches the problem of visually-guided reaching using connectionist method. MURPHY uses neural-like units to learn the association between visual perception and robot arm motions in which it learns both forward kinematics and inverse differential kinematics. To plan a collision free path , the system uses a heuristic to search the sequences of arm configurations which move the hand to the target. MURPHY moves the arm randomly and uses a gradient descent method to move closer to the target, backtracking if it fails and tries other paths if a current path is blocked by obstacles. Our system attempts to learn the similar task. The system learns the forward kinematics by

building a look-up table and solves motion planning part by using the genetic programming method.

Genetic programming (GP) is a machine learning technique derives from genetic algorithms (GA). Genetic algorithms, originated by Holland [5], are general-purpose search algorithm that use principles inspired by population genetics to evolve solutions to problems. GP [6] has become increasingly popular in recent years as a method for solving complex search problems in a large number of disciplines. Koza and Rice [7] applied genetic programming to generate robot programs that perform a box pushing task : a mobile robot searching for a box and pushes it to the wall. The applications of GA and GP to robot manipulator problems are numerous; Khoogar, Parker and Goldberg [8] solved inverse kinematics of redundant robots, Chan and Zalzala [9] planned minimum-time trajectories. 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 recognised [10], [11], [12]. This problem is especially interesting for us.

3: Learning the visual task

3.1: Problem statement

We define our visual task as follows: a robot arm reaches for a target while avoiding obstacles. The robot system consisted of a 3 degrees of freedom robot arm, which has all joints rotate on one plane, and a vision system (fig. 1). The target is specified as an object visible in the view of the vision system. Note that the task is specified in terms of visual objects and not by referring to any world coordinate system. The vision system can locate the target, the obstacles and the position of the arm, i.e. the locations of the shoulder, the elbow, the wrist and the fingertip. The vision system reports the locations in the image coordinate.

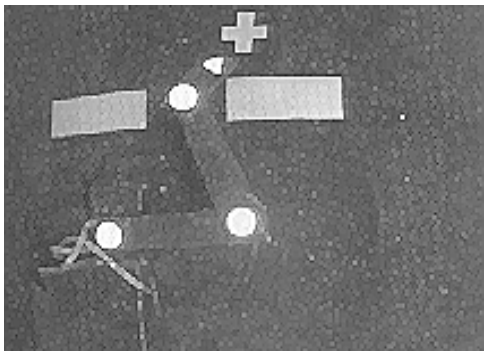


Fig. 1. A typical scene of the task

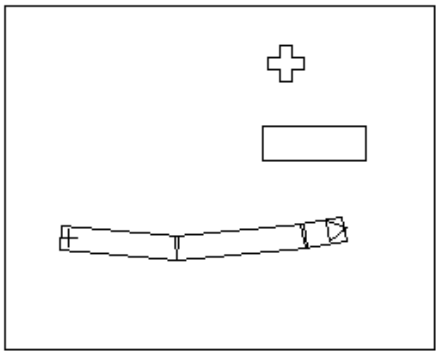
3.2: Learning process

The robot system described above is controlled by a robot program. The robot program is constructed from a list of commands with necessary functions to move the joints and to perceive the environment. The detailed description of a robot program is in the section 5. The learning process starts from a large number of randomly generated robot programs (400 programs in this experiment). Each program is evaluated by running it in the environment. Upon termination, the system observes the result how well the robot performs the task. From these results, the learning process tries to improve robot programs by altering their structures and commands, then the cycle of evaluating and improving is repeated until a solution is found. The solution is the robot program that can control the arm to reach the target while avoiding obstacles. The genetic programming method is used as the learning process in this paper and is discussed in details in the section 5. In the next section, we describe the design of the experiment and the robot system.

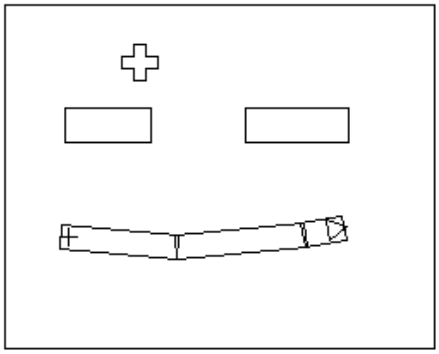
4: Experiments

We select three problems with different degree of difficulty for the robot system to learn (fig. 2). The first problem has one obstacle in front of the target. The robot has to fold the arm to move around the obstacle. The second problem puts more constraint on the possible trajectory of the arm by having two obstacles. The robot has to move through the opening to reach the target. The third problem creates a situation where a local minimum exists. There are two openings to the target but only the left one enables the robot to reach the target. If the robot enters the wrong opening, the arm will be wedged against the obstacles without reaching the target. Next we describe the set up of the robot system to run the experiment.

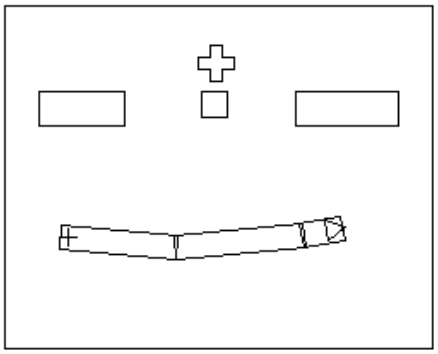
A vision system is used to identify the target, the obstacles and the robot arm. To simplify the vision algorithms, the robot arm has a distinctive color and two different thresholds are used to distinguish the robot arm from the obstacles and the target. A simple heuristic is used to locate three joints of the robot arm. The shoulder joint is the one near the bottom left of the image. The fingertip has a triangular shape. The wrist is the joint that is near to the fingertip and, lastly, the elbow is the remainder joint. The target, the obstacles and the arm are represented directly in the image therefore checking the collision and the out of bound condition are done with the actual image data. This is, in a way, similar to the analogical representation paradigm.



a) Problem 1



b) Problem 2



c) Problem 3

Fig. 2. Three problems to learn

To evaluate a large number of robot programs effectively, the real robot cannot be used as it takes an inordinate amount of time. Consider that we evaluate 4000 programs for each problem and run the experiment 20 times, a typical robot program takes about 100 sec. to run to completion on the real robot, so it will take about 2000 hours for each problem. Therefore, a simulator is used for this purpose. A robot program is evaluated by running it under simulated environment thus avoiding the speed limit of the real robot. The image of the environment (the arm, the target and the obstacles) that is used in the simulation run, is taken from the actual scene via the camera. The danger of using the simulator is that it might not correspond to the real run due to many factors. We discuss this problem in the discussion section.

One important element in the simulator is the forward kinematics model of the robot arm, i.e. the function that maps the robot joint angles to the position of links. We prefer to avoid using the mathematical model and instead use the look-up table that is constructed from the real data from the vision system. To construct the table of forward kinematics, the arm is moved, with a discrete incremental of joint angles, for every combination of joint angles, and the positions of the links are recorded by the vision system (the robot arm is controlled by joint motion commands such as *shoulder moves +5 degrees*). The position data are average over a number of repeated runs to smooth out the noise.

5: Genetic programming process

We describe the elements of a robot program and then the genetic programming process. The robot program has a tree structure, similar to the S-expression in LISP. The set of terminals includes six primitive servo motor functions and the system checking functions. Thus, the terminal set for this problem is { `s+, s-, e+, e-, w+, w-, hit?, see?, inc?, dec?, out?` }. A set of functions is {`if-and, if-or, if-not`}. Fig. 3 shows an example of a robot program.

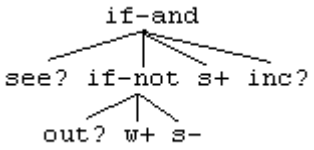


Fig. 3. An example of a robot program

The function `s+` (shoulder) drives the shoulder motor clockwise 1 step (5 degrees) and `s-` drives the shoulder motor anticlockwise 1 step. The similar meaning applies for `e+, e-` (elbow) and `w+, w-` (wrist). All of these functions always return true. The function `hit?` checks whether each link of the robot arm hits the obstacle. To discourage the action that causes repeated collisions a memory is created using PAIN-variable. When the arm hits an obstacle, 5 is added to the PAIN-variable, otherwise, 1 is subtracted from the PAIN-variable (lower limit at zero). The function `hit?` returns true if the PAIN-variable is not equal to zero, . The function `see?` checks whether the path from the fingertip to the goal has any obstacle. The function `inc?` checks whether the distance between the fingertip and the goal is increasing. The function `dec?` checks the opposite. The function `out?`

checks if each joint of the robot arm moves out of bound. The bound is defined to prevent the arm from going out of the view of the camera. The function `if-and` is a four-argument comparative branching operator that executes its third argument if its first argument and its second argument are true, or otherwise, executes the fourth argument. The function `if-or` is a four-argument operator that executes its third argument if its first argument or its second argument is true, or otherwise, executes the fourth argument. The function `if-not` is a three-argument operator that executes its second argument if the negation of its first argument is true, or otherwise, executes the third argument. We describe the genetic programming process next.

There are five stages in one cycle of the genetic programming process in the experiment.

Stage 1 : Creation of an initial population

The first step is to generate an initial population of computer program that randomly mixes the functions and the terminals. We use the size of population 400 programs. Each individual has at least 40 symbols. We check that there is no duplication in the initial population.

Stage 2 : Verification of each computer program

In each generation, iteratively execute each program until one of the termination criteria has been qualified. The termination criteria are described as follows:

- Maximum execution time : not over 100 execution time. The execution time is defined as the number of time that the program tree is evaluated.
- Dead condition : There are two conditions :
  1. The arm is in the same final position over 10 execution time, i.e. the arm gets stuck.
  2. The value of PAIN-variable is over 500 units.
- Successful : the arm has reached the target.

Stage 3 : Evaluation of each program

The fitness of individual program is evaluated with the following function :

$$2000 \times initialDistance / finalDistance + 100 \times sumDistance / finalDistance + 1000 \times notSee + 4000 \times die$$

where *initialDistance* is the initial distance from the fingertip to the target, *finalDistance* is the distance from the fingertip to the target after the program execution is terminated, *sumDistance* is the total of distance that the fingertip travelled, *notSee* is a boolean variable that indicate the path from the fingertip to the target is blocked by an obstacle, and *die* is a boolean variable that indicate the dead condition.

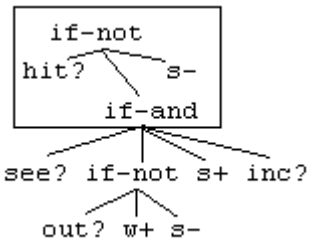
The fitness function is a cost function, the lower number means the better performance of the program. This function indicates that : 1) getting close to the target lower the cost 2) move economically has the lower cost 3) moving to an opening to the target has the lower cost and 4) penalise the badly performed program (die) by giving it a high cost.

Stage 4 : Selection of the good programs

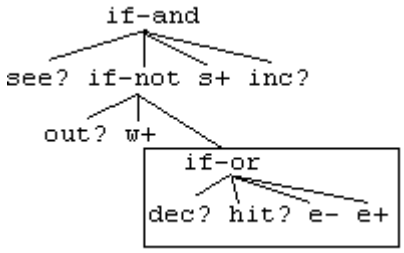
Based on the individual fitness of each program, the best 10% (40 programs) of population are selected to generate the population for the next generation.

Stage 5 : Genetic manipulation

The manipulation process uses genetic operators to create a new population of individual programs. There are four operations : reproduction, crossover, addition and extension. The reproduction is the operation that copies the best 40 programs into the next generation. The crossover is the operation that creates the new programs by recombining randomly chosen parts of two of the existing best 40 programs (the pair is chosen with replacement and enforces that they must be the different individuals). The addition operation generates an additional node (randomly generated) from the root node (fig. 4). The extension operations does likewise but from a terminal node. The addition and extension operations create the new programs that are longer than the originals and bring in the new part to the program population. The number of programs generated from crossover is 160, from addition 100 and from extension 100.



a) addition operation



b) extension operation

Fig. 4. The addition and extension operations

The number of generation is 10 for each experiment. Because of the stochastic nature of the method we need a number of runs to infer the result. We ran the genetic

programming process for the problem 1 and 2, 20 times. The problem 3 is more difficult to find solutions and we ran the experiment 40 times. We discuss the results in the next section.

6: Results and discussion

The results are presented in fig. 5, using the *performance curve* as defined by Koza in [13]. *P(M,i)* is the probability of a single run yielding a solution by *i* generations (each consisting of *M* individuals). This is estimated by doing a number of runs. The number of runs required to produce a successful individual with probability *z* is defined in terms of *P(M,i), R(z) = ceiling(log(1-z)/ log(1-P(M,i))).* The number of individuals that must be processed to find a successful individual with probability *z* is *I(M,i,z) = R(z).M.i.* The minimum of the *I(M,i,z)* is a measure of the difficulty of the problem, called *Effort (E).* We use the confidence factor *z* = 99%.

The results show that the robot system can learn to solve all problems satisfactorily. The problem 2 is easiest to solve which indicates that the environment actually *help* to guide the arm to move in the right direction. The problem 3 is hardest as expected, a lot of individual got stuck at local minima. We are interested in the *improvement* of the fitness of the population, in other words, the *learning* that took place during the genetic programming process. We plot the fitness value of a typical run (problem 1, run number 6, the first solution was found in the 6th generation) in fig. 6. There are two curves, one is the fitness of the best individual and another is the average fitness. The average fitness is calculated from the best 40 programs of that generation. One can notice that, although the initial population is randomly generated, some program perform better than the others and the genetic programming process *improves* the fitness of the population, eventually yields a solution. We also interested in the *quality* of the solution. In the fitness function, we give weight for the good quality solution, defined as the one that has a shorter distance trajectory (*sumDistance / initialDistance*). We plot the two trajectories in fig. 7, one is the trajectory of the best individual in the generation that the first solution is found, the other one is the trajectory of the best individual of the final (10th) generation from the same run (problem 1, run number 6). There is an improvement in the quality of the solution. Finally, we observe the correlation between the size of the solution, measured by the number symbols in the program, and the difficulty of the problem (*E*) in table 1. The size is calculated from the average size of the best 40 programs of the final generation (we select only the run that yield solutions). To give an idea about the time taken to run GP, we present some observation, one run (M = 400, generation = 10) takes about 20 sec. on the machine we used, a SPARCstation 20 with 4 CPUs of 50 MHz SuperSPARC, 192 Mbytes memory, during a normal load period (day time, 10 users on the machine).

a) Problem 1

b) Problem 2

c) Problem 3

Fig. 5. The performance curves of the three problems

Fig. 6. The learning curve of the problem 1, run number 6

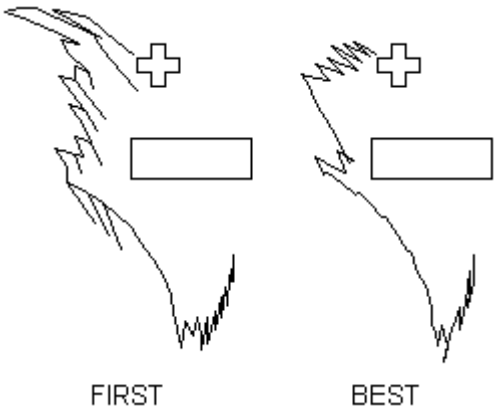


Fig. 7. The improvement of trajectory (from the problem 1, run number 6)

Table 1. The correlation between size of solution and the effort

	p. 1	p. 2	p. 3
average size (symbols)	42	36	66
effort (no. of individual)	16000	2800	72000

To validate solutions, we select 20 programs from the best individual of each run of problem 1 and problem 2, 5 programs from the problem 3, and run them with the real robot. The result is presented in table 2. This result compares well with the result from [15] which used the same equipment. The reason for failure is due to the uncertainty in the actual runs. Although the result is satisfactory, it is not *robust*. The initial condition must be exactly the same as in the simulation, even a small deviation can lead to failure to reach the target. This has been observed in [10] that a major difficulty in using the learning techniques that based on simulation lies in the transferring of programs evolved in simulated environments to actual robots. To use the robot in the real world, Dorigo's work [11] suggests that the learning process should be taken place in the actual run. Reynolds [12] tries to simulate the uncertainty by injection noise into the simulation by has no success in getting solutions. Ito, Iba and Kimura [14] study the robustness of robot programs generated by GP. Their chosen task is a box moving problem. A mobile robot moves a box to the target location. The robustness is studied in two aspects : 1) the initial position of robot is random 2) the noise is injected into sensors and actuators. Under the simulation, their results show that the generated robot programs are robust. In our work, we have tried as much as possible to emulate the real world in our simulation by using the actual visual data and the forward kinematics from the real data.

Table 2. The number of generated programs that successfully run with the real robot

	[15]	this paper
problem 1	80%	90%
problem 2	70%	90%
problem 3	83%	80%

The work in [15] as well as our earlier work [16] use the forward kinematics model in the simulator. We found that the run on the actual robot is not the same as the run in the simulation. This is caused by the discrepancy in position predicted by the kinematics model and the actual position of the robot arm. This error comes from several sources : the non-linearity of the camera field of view, the error in the image processing, the error in servo motors. We ran a test to see these cumulative effects. The result is presented in fig. 8. The graph shows the length of the last link of the arm (the finger) when the robot is in various configurations compare to the average length (averaging the length in all configurations). The size of the icon represents the difference (scale by 2). One can observe that, from the view of the vision system, the top-left corner appears compressed and the middle region appears enlarged. The distortion is highly non-linear. The result of this test convinced us that it is unlikely that we can find a mathematical model that can predict the position of the links from the joint angles with adequate accuracy. Therefore we adopted the look-up table approach.

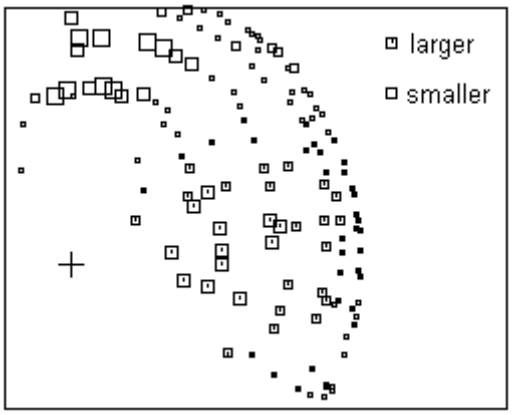


Fig. 8. The cumulative error effect

7: Conclusion and future work

This work shows that the genetic programming method can be used to solve the motion planning problem in a visually-guided reaching task. The system improves its performance by learning from its experience. The vision system is used to map the environment directly into an internal representation. This representation is use successfully as a simulated environment for genetic programming. The result from the experiment indicates that the solution is not yet *robust* when applies to the real robot. More work can be done in this area. One approach that we are pursuing is the use the real robot to learn in the real world. To overcome the limitation of the speed of the robot arm, we are investigating the on-line algorithm that dynamically build the forward kinematics and the map of the environment at the same time as the learning is progressing.

Acknowledgements

We would like to thank the department of computer science, Michigan State University, for providing the environment and the computational resource to carry out this work and the department of computer engineering, Chulalongkorn University, for permitting an extended visit of the first author to Michigan State University. The second author is supported by the grant from the National Science and Technology Development Agency of Thailand. We also thank the reviewers whose comments had helped us to improve the clarity of this paper.

References

[1] V. Jones, “Tracking: An Approach to Dynamic Vision and Hand-Eye Coordination,” Ph.D thesis. University of Illinois, Urbana Champaign, 1974.

[2] T. Lozano-Perez, J. L. Jones, E. Mazer and P. A. O'Donnell, *HANDEY A Robot Task Planner*. MIT Press, 1992.

[3] S. A. Hutchinson and A. C. Kak, “Spar: A planner that satisfies geometric goals in uncertain environments”, *AI Magazine*, vol. 11, no. 1, 1990, pp. 30-61.

[4] B. W. Mel, *Connectionist Robot Motion Planning: A Neurally-Inspired Approach to Visually-Guided Reaching*. Academic Press, 1990.

[5] J. H. Holland, *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: University of Michigan Press, 1975.

[6] J. R. Koza, *Genetic Programming*. MIT Press, 1992.

[7] J. R. Koza and J. P. Rice, “Automatic Programming of Robots using Genetic Programming,” in *AAAI-92 Proc. Tenth National Conf. on Artificial Intelligence*, pp. 194-201.

[8] A. R. Khoogar, J. K. Parker and D. E. Goldberg, “Inverse Kinematics of Redundant Robots using Genetic Algorithms,” in *Proc. IEEE Int. Conf. on Robotics and Automation*, vol. 1, 1989, pp. 271-276.

[9] K. K. Chan and A. M.S. Zalzala, “Genetic-Based Minimum-Time Trajectory Planning of Articulated Manipulations with Torque Constraints,” *IEE Colloquium on Genetic Algorithms for Control Systems Engineering*, Digest No. 1993/130, 1993, pp. 4/1-3.

[10] R. A. Brooks, “Artificial Life to actual robots”, in *Proc. of the first European conf. on Artificial Life*, MIT Press, 1991, pp.3-10.

[11] C. W. Reynolds, “Evolution of obstacles avoidance behavior: using noise to promote robust solutions”, in K. Kinnear, Ed., *Advances in genetic programming*. MIT Press, 1994.

[12] M. Dorigo, “ALECSYS and the AutonoMouse: Learning to control a real robot by distributed classifier systems”, *Machine learning*, vol. 19, 1995, pp.209-240.

[13] J. R. Koza, *Genetic Programming II*. MIT Press, 1994, pp. 99-105.

[14] T. Ito, H. Iba and M. Kimura, “Robustness of robot programs generated by Genetic Programming”, In *Genetic Programming 96*, MIT Press, 1996.

[15] J. Polvichai, “Robot Learning by Genetic Programming”, M.Eng thesis, the department of computer engineering, Chulalongkorn University, Thailand, 1996 (in Thai).

[16] J. Polvichai and P. Chongstitvatana, "Visually-Guided Reaching by Genetic Programming", in *Proc. of ACCV'95 the second Asian Conf. on Computer Vision*, vol. 3, 1995, pp. 329-333.