Researches on Ingeniously Behaving Agents

Shotaro Kamio Hongwei Liu Hideyuki Mitsuhasi Hitoshi Iba

Graduate School of Frontier Science, The University of Tokyo,
7-3-1 Hongo, Bunkyo-ku, Tokyo, 113-8656, Japan.
{kamio,lhw,mituhasi,iba}@miv.t.u-tokyo.ac.jp

Abstract

We have been studying the techniques for evolutionary robotics and experimenting with various robots applied evolutionary methods. We have paid special attentions to real robots and multi-agent problems with them. In this research domain, we named them as "Ingeniously Behaving Agents (IBA)". This paper shows several techniques developed in our IBA laboratory and their experimental results applied to simulations and real robots.

1. Introduction

We have been studying the techniques for evolutionary robotics by using Genetic Algorithms and Genetic Programming. Our goal is to clarify the applicability of the evolutionary approach to the real-robot learning, especially, in view of the multi-agent cooperation, the adaptive behavior and the robustness to noisy and dynamic environments. For this purpose, we use a variety of real robots called "Ingeniously Behaving Agents (IBA)" as follows:

(a) AIBO (Fig. 5): An entertainment four-legged robot by SONY.

(b) Khepera K-TEAM (Fig. 7): A set of miniature mobile robots of K-Team S.A.

(c) HOAP-1 (Fig. 14): A miniature humanoid robot by Fujitsu Automation Limited.

Next sections show techniques developed in our IBA laboratory and their experimental results.

2 Real-time adaptive technique to real robots by means of integration of GP and RL

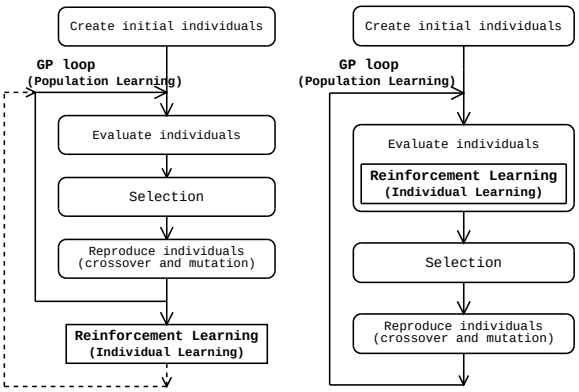
When executing tasks by autonomous robots, we can make the robot learn what to do so as to complete the task from interactions with its environment but not manually pre-program for all situations. We know that such learning techniques as genetic programming (GP)[11] and reinforcement learning (RL)[15] work as means for automatically generating robot programs.

When applying GP, we should repeatedly evaluate many individuals over several generations. Therefore, it is difficult to apply GP to problems that requires too much time for evaluations of individuals. That is why we find very few previous studies on learning with a real robot.

To obtain optimal actions using RL, it is necessary to repeat learning trials time after time. The huge amount of learning time required presents a great problem when using a real robot. Accordingly, most studies deal with the problems of receiving an immediate reward from an action as shown in [9], or loading the results learned with a simulator into a real robot as shown in [1, 16].

Although it is generally accepted to learn with a simulator and apply the result to a real robot, there are many tasks that are difficult to make a precise simulator. Applying these methods with an imprecise simulator could result in creating programs which may function optimally on the simulator but cannot provide optimal actions with a real robot. Furthermore, the operating characteristics of a real robot show certain variations due to minor errors in the manufacturing process or to changes with time. We cannot cope with such differences of robots only using a simulator.

To solve the above difficulties, we have proposed a technique that allows a real robot to execute real-time learning in which GP and RL are integrated[8]. Our proposed technique does not need a precise simulator because learning is done with a real robot. As a result of this idea, we can greatly reduce the cost to make the simulator much precise and acquire the program which acts optimally in the real robot. Moreover, learning with a real robot sometimes makes possible to learn even hardware and environmental characteristics, thus allowing the robot to acquire unexpected actions.



(a) Proposed technique of the integration of GP and RL.

(b) Traditional method combined GP and RL [6, 3].

Figure 1. The flow of the algorithm.

2.1 Task definition

We used an "AIBO ERS-220" robot sold by SONY as the real robot in this experiment. AIBO's development environment is freely available for non-commercial use and we can program with C++ language on it [13]. An AIBO is equipped with a CCD camera on its head.

The task in this experiment was to carry a box to a goal area. One of the difficulties of this task is that the robot has four legs. As a result, when the robot moves ahead, we see cases where the box sometimes is moved ahead or deviates from side to side, depending on the physical relationship between the box and AIBO legs. It is extremely difficult, in fact, to create a precise simulator that accurately expresses this box movements.

2.2 Proposed technique

As can be seen in Fig. 1(a), RL is outside of the GP loop in the proposed technique. This technique enables us (1) to speed up learning in real robot and (2) to cope with the differences between a simulator and a real robot.

The proposed technique consists of two stages (GP part and RL part).

1. Carry out GP on a simplified simulator, and formulate programs that have the standards for robot actions required for executing a task.
2. Conduct RL after loading the programs obtained in Step 1 above.

The learning process of RL can be speeded up in the second step because the state space is divided into partial spaces under the judgment standards obtained in the first step. Moreover, preliminary learning with a simulator allows us to anticipate that a robot performs target-oriented actions from the beginning of the second stage. We used Q-learning as RL method in this study.

Although the process expressed by the external dotted line in Fig.1(a) was not realized in this study, it is a feedback loop. We consider that the parameters in a real environment that have been acquired via individual learning should ideally be fed back through this loop route.

We see several studies in which GP and Q-learning are combined as Fig. 1(b) [6, 3]. However, no studies using any of them with a real robot have been reported because these techniques must execute Q-learning for numerous individuals in the population,

2.2.1 RL part conducted on the real robot

Action set We prepared six selectable robot actions (move forward, retreat, turn left, turn right, retreat + turn left, and retreat + turn right). These actions are far from ideal ones: e.g. "move forward" action is not only to move the robot straightly forward but also has some deviations from side to side and "turn left" action is not only to turn left but also move the robot a little bit forward. The robot has to learn these characteristics of actions.

State Space The state space was structured based on positions from where the box and the goal area can be seen in the CCD image, as described in [1]. We added a mechanism to compensate for the surrounding images by swinging AIBO's head so that state recognition can be conducted after each action.

Figure 2 is the projection of the box state on the ground surface. The "near center" position is where the box fits into the two front legs. The box can be moved if the robot pushes it forward in this state. The box remains same position "near center" after the robot turns left or right in this state because the robot holds the box between two front legs. The state with the box not being in view was defined as "lost"; the state with the box not being in view and one preceding step at the left was defined as "lost into left" and, similarly, "lost into right" was defined.

We thus defined 14 states for the box. We similarly defined states of the goal area except that "near straight left" and "near straight right" states do not exist in them. There are 14 states for the box and 12 for the goal area; hence, this environment has states of their product, i.e., 168 states totally.

2.2.2 GP part conducted on the simulated robot

Simulator The simulator in our experiment uses a robot expressed in circle on a two-dimensional plane, a box, and a goal area fixed on a plane. The task is completed when the robot pushes the box forward and overlaps the goal area on this plane.

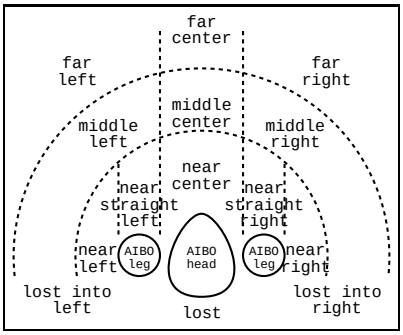


Figure 2. States in real robot for the box. The front of the robot is upside of this figure.

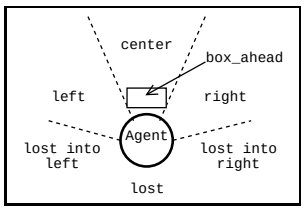


Figure 3. States for box and goal area in the simulator. The area `box_ahead` is not the state but the place where `if_box_ahead` executes first argument.

We defined three actions (move forward, turn left, turn right) as action set and defined the state space in the simulator as Fig. 3. While actions of the real robot are not ideal ones, these actions in the simulator are ideal ones.

Such actions and a state division are similar to that of a real robot, but are not exactly the same. In addition, physical parameters such as box weight and friction were not measured nor was the shape of the robot taken into account. Therefore, this simulator is very simple and it is possible to build it in low cost.

The two transfer characteristics of the box expressed by the simulator are the followings:

1. The box moves forward if the box comes in contact with the front of the robot when the robot goes ahead[1].

2. After rotation, the box is near the center of the robot if the box is near the center of the robot when the robot turns[2].

Settings of GP For executing GP in the simulator, we used terminal set = { `move_forward`, `turn_left`, `turn_right` } and function set = { `if_box_ahead`, `box_where`, `goal_where`, `prog2` }. The terminal nodes respectively correspond to the "move forward", "turn left", and "turn right" actions in the simulator. The functional nodes `box_where` and `goal_where` are the functions of six arguments, and they execute one of the six arguments, depending on the states (Fig. 3) of the box and the goal area as seen by the robot's eyes. The function `if_box_ahead` which has two arguments executes the first argument if the box is positioned at "box ahead" position in Fig. 3. We arranged conditions so that only the `box_where` or the `goal_where` node becomes the head node of a gene of GP. The gene of GP is set to start executing from the head node and the execution is repeated again from the head node if the execution runs over the last leaf node until reaches maximum steps.

A trial starts with the state in which the robot and the box are randomly placed at the initial positions, and ends when the box is placed in the goal area or after a predetermined number of actions are performed by the robot. To make robot acquire robust actions that do not depend on the initial position, the average values of 100 trials in which the initial position is randomly changed was taken when calculating the fitness of individuals (see [8] for the definition of the fitness).

Learning was executed for 1,000 individuals of 50 generations with maximum gene length $= 150$. Learning costs about 10 minutes on the Linux system equipped with an Athlon XP 1800+. We finally applied the individuals that had proven to have the best performance to learning with a real robot.

2.2.3 Integration of GP and RL

Q-learning is executed to adapt actions acquired via GP to the operating characteristics of a real robot. This is aimed at revising the `move_forward`, `turn_left` and `turn_right` actions with the simulator to their optimal actions in a real world.

We allocated a Q-table, on which Q-values were listed, to each of the `move_forward`, `turn_left` and `turn_right` action nodes. The states on the Q-tables are regarded as those for a real robot. Therefore, actual actions selected with Q-tables can vary depending on the state, even if the same action nodes are executed by a real robot. Figure 4 illustrates the above situation.

Each Q-table is arranged to set the limits of selectable actions. This refers to the idea that, for example, "turn right" actions are not necessary to learn in the `turn_left` node. In this study, we defined three selectable robot actions for each action node as Table 1. With this technique, each Q-table was initialized with a biased initial value[3]. The ini-

[1]This corresponds to the situation that real robot pushes the box forward.

[2]This corresponds to the situation in which the box is placed between the front legs of a real robot when it is turning.

[3]According to the theory, we can initialize Q-values with arbitrary values, and Q-values converge with the optimum solution regardless of the initial value [15].

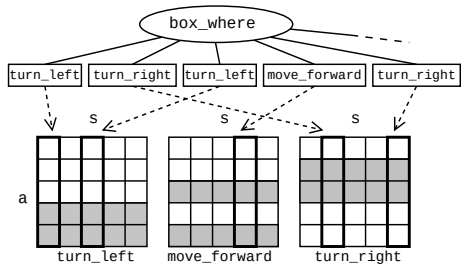


Figure 4. Action nodes pick up a real action according to the Q-value of a real robot's state.

tial value of 0.0001 was entered into the respective Q-tables so that preferred actions were selected for each Q-table, while 0.0 was entered for other actions. The actions which are preferred to select on each action node are described in Table 1.

The "state-action deviation" problem should be taken into account when executing Q-learning with the state constructed from a visual image [1]. This is the problem that optimal actions cannot be achieved due to the dispersion of state transitions because the state composed only of the images remains the same without clearly distinguishing differences in image values. To avoid this problem, we redefined "changes" in states. The redefinition is that the current state is unchanged if the terminal node executed in the program remains the same and so does the executing state of a real robot[4]. Until the current state changes, the Q-value is not updated and the same action is repeated.

As for parameters for Q-learning, the reward was set at 1.0 when the goal is achieved and 0.0 for other states. We set the parameters as the learning rate $\alpha = 0.3$ and the discount factor $\gamma = 0.9$.

2.3 Experimental results with AIBO

Just after starting learning: The robot succeeded in completing the task when Q-learning with a real robot started using this technique. This was because the robot could perform actions by taking advantage of the results learned via GP.

At the situation in which the box was placed near the center of the robot along with robot movements, the robot always achieved the task with regard to all the states tried. Whereas, if the box was not placed near the center of the robot after its displacement (e.g. if the box was slightly outside the legs), the robot sometimes failed to move the box properly. The robot repeatedly turned right to face the box, but continued vain movements going around the box because it did not have a small turning circle, unlike the actions in the simulator. Figure 5(a) shows typical series of actions. In some situation, the robot turned right but could not face the box and lost it in view (at the last of Fig. 5(a)).

This typical example proves that optimal actions with the simulator are not always optimal in a real environment. This is because of differences between the simulator and the real robot.

After ten hours (after about 4000 steps): We observed optimal actions as Fig. 5(b). The robot selected "retreat" or "retreat + turn" action in the situations in which it could not complete the task at the beginning of Q-learning. As a result, the robot could face the box and pushed the box forward to the goal, and finally completed the task.

Learning effects were found in other point, too. As the robot approached the box smoothly, the number of occurrence of "lost" was reduced. This means the robot acts more efficiently than the beginning of learning.

2.4 Comparison with Q-learning in both simulator and real robot

We compared our proposed technique with the method of Q-learning which learns in a simulator and re-learns in a real world (we call this method as RL+RL in this section). For Q-learning in the simulator, we introduced the qualitative distance ("far", "middle", and "near") so that the state space could be similar to the one for the real robot[5].

For this comparison, we randomly selected ten situations which are difficult to complete at the beginning of Q-learning because of the gap between the simulation and the real robot. We measured action efficiency after ten-hour Q-learning for these ten situations. These tests are executed in a greedy policy in order that the robot always selects the best action in each state.

Table 2 shows the result of both methods, i.e., proposed technique (GP+RL) and Q-learning method (RL+RL). This table represents the average number of steps to complete the task and the number of occurrences when the robot has lost the box or the goal area in completing the task. While RL+RL performed better than the proposed technique in four situations on the average of the steps, the proposed technique performed much better than RL+RL in other six situations (bold font in Table 2). Moreover, the robot evolved by the proposed technique less often lost the box and the goal area than that by RL+RL. This result proves that our proposed technique learned more efficient actions than RL+RL method.

Figure 6 shows the changes in Q-values when they are updated in Q-learning with the real robot. The absolute

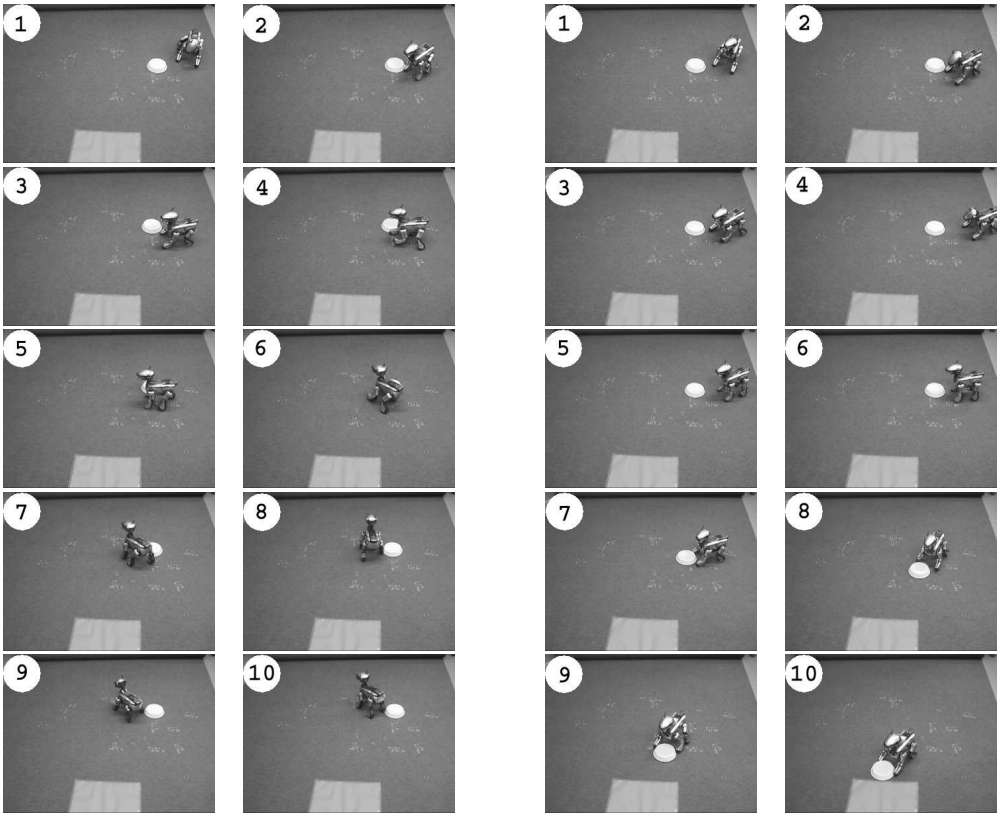
[4]We modified Asada et al.'s definition [1] in order to deal with several Q-tables.

[5]This simulator has 12 states for each of the box and the goal area; hence, this environment has 144 states.

Table 1. Action nodes and their selectable real actions.

action node	real actions which Q-table can select.
`move_forward`	"move forward"*, "retreat + turn left", "retreat + turn right"
`turn_left`	"turn left"*, "retreat + turn left", "retreat"
`turn_right`	"turn right"*, "retreat + turn right", "retreat"

* The action which Q-table prefers to select with a biased initial value.



(a) Failed actions losing the box at the beginning of learning.

(b) Successful actions after 10-hour learning.

Figure 5. Typical series of actions.

value of the Q-value change represents how far the Q-value is from the optimal one. According to Fig. 6, large changes occurred to RL+RL method more frequently than to our technique. This may be because RL+RL has to re-learn optimal Q-values starting from the ones which have already been learned with the simulator. Therefore, we can conclude that RL+RL requires more time to converge to optimal Q-values.

3 Multi-agent learning of heterogeneous robots by evolutionary subsumption

3.1 Evolutionary subsumption

In a multi-robot system several robots simultaneously work to achieve a common goal via interaction; their behaviors can only emerge as a result of evolution and interaction. How to learn such behaviors is a central issue of Distributed Artificial Intelligence, which has recently attracted much attention[4, 5, 7, 17, 18, 19]. We address the issue in the context of a heterogeneous multi-robot system, in which two real robots, i.e., Khepera, are evolved using GP to solve a cooperative task. Since directly using GP to generate a program of complex behaviors is difficult, a

Table 2. Comparison of proposed technique (GP+RL) with Q-learning (RL+RL).

#. situation	GP+RL			RL+RL		
	avg. steps	lost box	lost goal	avg. steps	lost box	lost goal
1	**19.6**	0	1	20.0	0	1
2	**14.7**	0	0	53.0	2	2
3	**24.0**	0	1	26.7	0	1
4	**10.3**	0	0	11.0	0	0
5	**21.6**	0	0	88.0	3	3
6	13.5	0	0	**10.5**	0	0
7	26.7	0	1	**26.0**	0	1
8	23.0	0	1	**13.0**	0	0
9	21.5	0	0	**10.5**	0	0
10	**13.5**	0	0	29.0	0	1

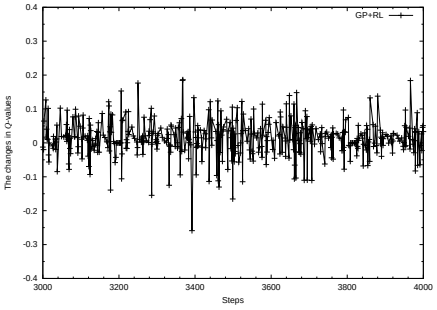
number of extensions to basic GP have been proposed to solve these control problems of the robot. For instance, J. Koza employed GP to generate a subsumption architecture control program[10]. W. F. Punch et al. proposed an approach to solve robot navigation problems, it incorporated subsumption principles into the Echo Augmented Genetic Programming approach[14].

In our previous researches we studied the emergence of the cooperative behavior in multiple robots/agents by means of GP and proposed three types of strategies, i.e., homogeneous breeding, heterogeneous breeding, and coevolutionary breeding, for the purpose of evolving the cooperative behavior[5]. We used a heterogeneous breeding approach of GP, evolving a multi-agent learning system, to solve robot navigation and Tile World problems[7]. We also applied the proposed GP system to a homogeneous cooperative multi-robot system and tested our approach in an "escape problem"[18]. These researches showed that GP is efficient in multi-robot/agent learning.

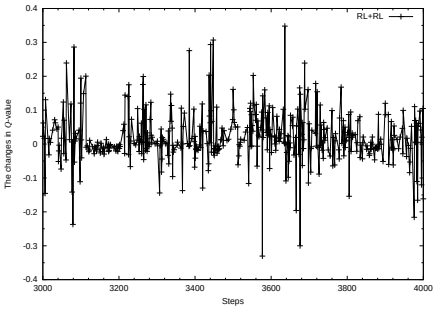
In this section, We report an improvement of GP, called Evolutionary Subsumption – which combines the GP with Brooks' subsumption architecture[2], and test our approach with an "eye"-"hand" cooperation problem (refer to [12] for more details).

3.2 "Eye"-"hand" cooperation problem

In this task two heterogeneous robots learn complex robotic behaviors by cooperation. One of them, which is mounted with a digital camera, acts as the "eye" and the other, which is mounted with a gripper, acts as the "hand" (Fig. 7). Their task is: the "eye" tries to find a cylindrical object and then navigates the "hand" to pick it up and then navigates it to carry the cylinder to the goal. The two robots are heterogeneous–they have different sensors and actuators, and have different roles in the system. Their behaviors are complex: including tracking, path planning, and



(a) Proposed technique (GP+RL).



(b) Q-learning (RL+RL).

Figure 6. Comparison of changes in Q-values after about 8-hour to 10-hour Q-learning with a real robot.

communication, etc.

In this system, since the relative position between "eye" and "hand" is variable, the "eye" must track two objects– "hand" and cylinder simultaneously, moreover one of the two objects–the "hand" is movable, the search space of our target system is large and the emergence of robots' rational strategies is very difficult.

The "eye" must select suitable viewpoints, observe the environment, and send correct instructions to "hand". Along with the moving of the "hand", the "eye" must be able to adjust its position and send new instructions according to the new situation.

3.3 Methodology

3.3.1 Design of architecture

According to analysis in Sect. 3.2, search space of our target system is very large, it is intractable to search for a direct solution using Genetic Programming. The divide-and-conquer approach is an intuitive and efficient method when we encounter complex problems. Being a divide-and-conquer approach, the subsumption architecture decomposes the problem into a set of levels[2] and each level

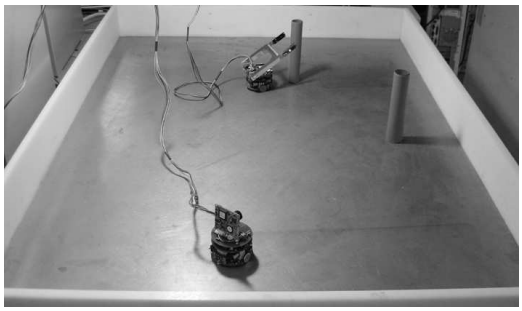


Figure 7. "Eye"-"hand" cooperation problem.

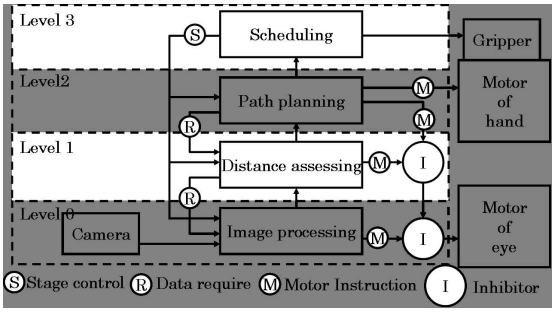


Figure 8. Evolutionary subsumption approach's layered architecture.

implements a task-achieving behavior. We employed the subsumption architecture, dividing the whole behavior into several simple behaviors. Then each level is automatically generated by Genetic Programming respectively; the lower level is formed by Genetic Programming at first, and then uses lower levels' output as nodes of the next level of Genetic Programming.

3.3.2 Evolutionary subsumption

The control system is divided into 4 levels: level0 image processing, level1 distance assessing, level2 path planning, and level3, scheduling. See Fig. reffig:architecture. The rest of this section will introduce each level of the architecture.

Level0 image processing This level gets an input image, detects whether the "hand" and cylinders appear in the view or not, and calculates the width of their image. In order to fix our attention on the task of coordination and not immerse ourselves in the field of machine vision, we use particular colors to identify the "hand" and cylinder. See Fig. 9, input at this level is one scan line of the image and outputs are W_{hand}, D_{hand}, W_{obj}, D_{obj} (i.e., the offset from center

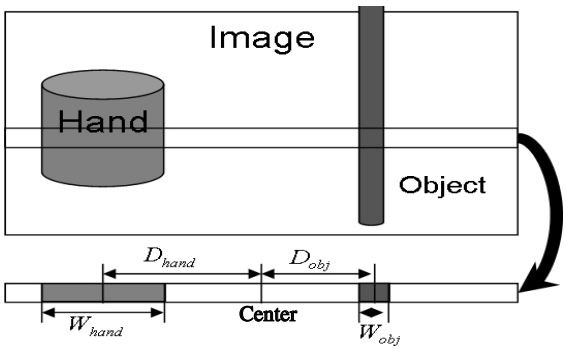


Figure 9. Image processing approach of level0

of image and the width), and two Boolean variables B_{hand} and B_{obj}, they indicate whether the"hand" and the cylinder are within the image.

Level1 distance assessing Level1 takes level0's output as its input and assesses the distance of "eye"–"hand" and "eye"–cylinder. Therefore the task of level1 is a symbolic regression problem:

$$f(W_{hand}, D_{hand}, W_{obj}, D_{obj}, B_{hand}, B_{obj}) = \{Dis_{hand}, V_{hand}, Dis_{obj}, V_{obj}\} \quad (1)$$

Where Dis_{hand} and Dis_{obj} are the assessed distances, and V_{hand} and V_{obj} indicate whether the assessed distance is valid or not. These values will be used by the higher levels. If the objects, i.e., the "hand" and cylinder, do not appear in view-field of "eye" or are too far from the "eye" then V_{hand} and V_{obj} will be set to "False", otherwise they will be set to "True".

This level is trained separately. For each generation, before training we generate 10 maps, which randomly specify the position and orientation of the "eye", the "hand", and the cylinder. These 10 maps will be kept constant within one generation; in the next generation they will be reformed, i.e., will be different from the prior generations. The fitness is defined as the average error between the assessed value and the real value in the 10 maps. Figure 10 shows the best fitness of this level, we can observe that along with the evolution, the assessing distance is getting more and more accurate.

Level2 path planning The task of this level is to generate rational motor instructions. In our approach we used central-control architecture. It generates motor instructions for both "eye" and "hand". The rational instructions for the "eye" are to get a better viewpoint and the rational instructions for the "hand" are to drive it closer to the cylinder. As we will observe in Sect. 3.3 the two robots will learn to coordinate with each other gradually and the rational strategy

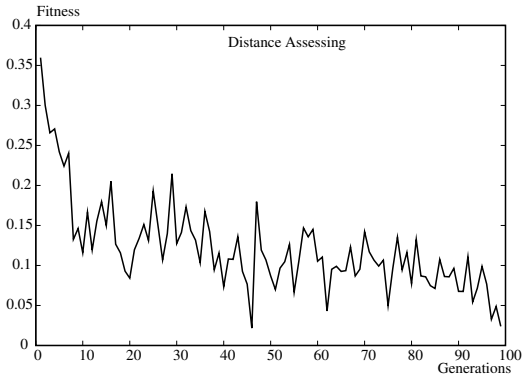


Figure 10. Fitness of distance assessing of level1.

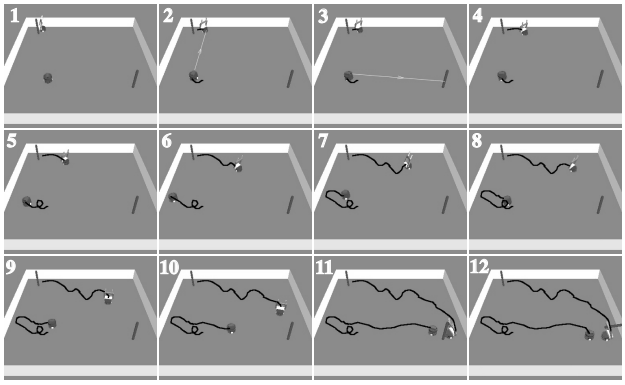


Figure 11. The skillful cooperative behaviors emerged.

will emerge along with the evolutionary procedure.
Level3 scheduling This level determines when the "hand" should pick up the cylinder and when it should put the cylinder down. Since the procedure of this level is fixed, we can write the program for this level manually.

3.4 Experiments with evolutionary subsumption

3.4.1 Environment and experimental setting

We used Webots of Cyberbotics for the experiments. The size of the environment is 100×100 cm with high 10 cm white walls, so that the "eye" can recognize "hand" and cylinder easily. In order to keep things simple we used special colors to identify the cylinder and the goal. The "eye" robot was equipped with a k6300 digital camera turret and the "hand" robot was equipped with a Gripper turret. The initial positions of "eye", "hand", cylinder, and goal are placed randomly. The limit of steps is 2 times the linear distance between "hand" and cylinder. The actions of a robot are simplified to 4 actions: MF move forward, MB move backward, MR turn right about 30 degree and move forward, and ML turn left about 30 degree and move forward.
At the beginning of each generation we generate maps by randomly placing the "eye", "hand", and cylinders. These maps are kept constant only within one generation. They will be regenerated before evolution to the next generation. The number of maps, i.e., the fitness cases, increase with the generations from 1 to 10. That is, along with evolution the difficulty of the task is increased, finally each individual must be evaluated on 10 maps. This method is able to prevent the robots from accomplishing their task by fluke. Fitness is defined as the maximum distance in all fitness cases and the distance is between the "hand" and the cylinder after the "hand" runs out of its steps.

We used function set F={IFLTE, PROGN2, Data_req, IFVhand, IFVobj} and terminal set T={D_{hand}, D_{obj}, Scan, MFe, MBe, MLe, MRe, MFh, MBh, MLh, MRh}, where the function set is very similar to level1; in the terminal set D_{hand} and D_{obj} are the output of level1, the MFe, MBe, MLe, MRe are the motor instructions of "eye", the others are the motor instructions of "hand". The other parameters are population size: 2000; crossover rate: 0.85; mutation rate and elite rate: 0.1; maximum depth: 15.

3.4.2 Result

At the beginning of evolution the two robots show poor coordination. Usually the "eye" and the "hand" move separately; the "hand" moves aimlessly before the "eye" surveys the environment and soon it runs out of its steps unnecessarily. Even in generation 0, there are some individuals better than others, they approach the cylinder more closely.
Along with the evolution the two robots gain more skill in cooperation, they show clear rhythm of "observation"–"action"–"observation"..... The "hand" never moves before the "eye" because it must save its limited steps.
Finally, the two robots are more skillful. They have averaged more than 60% probability to accomplish the task. Figure 11 shows their trajectory. As shown in Fig. 11, the two robots show favorable coordination. At first the "eye" observes the environment and directs the "hand" to move and then the "eye" observes again adjusting its position and directing the "hand" to move again... We can also observe that the trajectory of "hand" is getting more and more smooth along with their interaction. These phenomena indicate that the rational strategy has emerged.
For comparison, we also employed direct GP approaches to solve the problem. In the direct GP approach, in order to keep things simple, we did not use the input image directly; instead, we kept the level0 fixed and just used GP to gener-

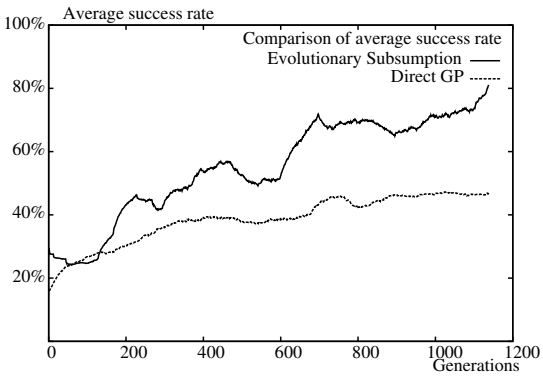


Figure 12. Comparison of averaged success rates (10 runs) of evolutionary subsumption and direct GP.

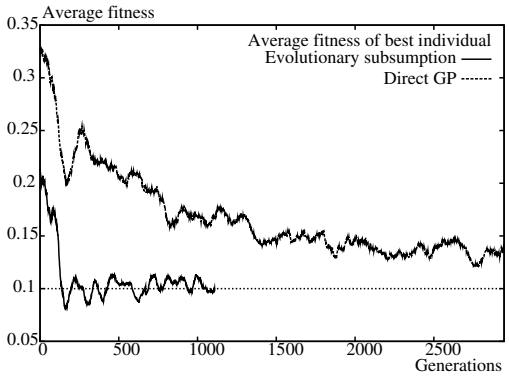


Figure 13. Comparison of the best individual's averaged fitness (10 runs) of the direct GP approach and the evolutionary subsumption approach.

ate programs for level1 and level2. The definition of fitness and the other parameters are the same as in the evolutionary subsumption approach. In direct GP approach, although the "eye" and the "hand" can find the rational strategies and produce cooperative behaviors, they take almost 3 times the number of generations of the evolutionary subsumption approach and often failed to converge due to premature convergence.

Figure 13 shows the comparison of the best individual's fitness over the generations and in Fig. 12, we can find the success rate of the evolutionary subsumption approach superior to the direct GP approach. This seems due to the reasonable subsumption architecture, we have designed the suitable framework for the whole system and the GP need only search the optimal solution for each layer in a relatively small search space. On the other hand, the direct GP approach has to search in a large search space and often times it can not, or it must take a number of generations' evolution to decompose the problem into rational components. Therefore, the final results are inferior to the subsumption architecture.

4 Applications to humanoid robots

To confirm the real-world effectiveness, we have applied the above-mentioned techniques to humanoid robots. These robots are of 20 DOF's (Degree Of Freedom) and are equipped with a CCD camera.

We conducted an experiment with the same task under almost the same condition as described in Sect. 2.1. Note that the simulator (GP part) in Sect. 2.2 assumed no particular specification of the robot. Thus, we can use the proposed technique for the sake of experimenting with a humanoid robot as well, provided that the real-robot learning (RL part) is performed in a real humanoid situation. We have defined five actions (i.e., move forward, turn left, turn right, side-step left, side-step right) as an action set of a humanoid robot. A new state space was introduced for the Q-learning with the real robot. As a result of experiments, the robot succeeded in completing the task. The evolved humanoid was able to move a target to a goal by kicking it (Fig. 14). Its behavior was similar to the one acquired by AIBO in Sect. 2, but not identical because of the real-robot learning.

5 Conclusions and future works

We presented techniques and experimental results which have been pursued in our laboratory for evolutionary robotics. We can conclude the following points as to the applicability of our evolutionary approach:

- The integrated technique of GP and RL makes possible to acquire optimal actions with a real robot in real-time. The comparison result showed that this technique is more efficient than traditional Q-learning method. As the learning is done with a real robot, we can make the simulator much precise.
- Evolutionary Subsumption is efficient in emergence of a heterogeneous multi-robot system. It shows superiority to both classical subsumption architecture and GP approach. As a divide-and-conquer method evolutionary subsumption approach can integrate designer's knowledge with artificial evolutionary approach, restrict the search space of artificial evolution approach, and thus improve the performance of evolution approach.

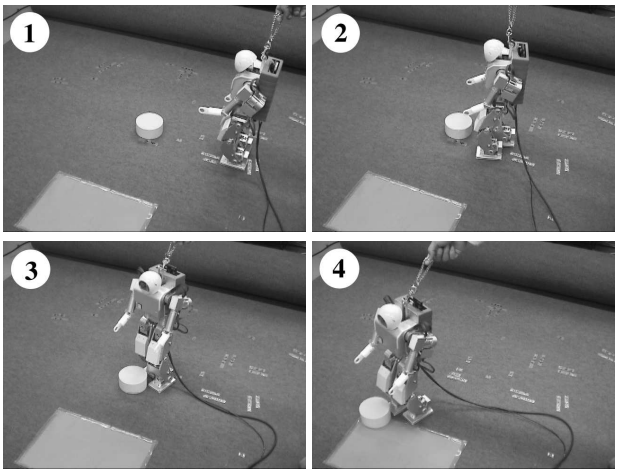


Figure 14. The integrated technique of GP and RL applied to a humanoid robot.

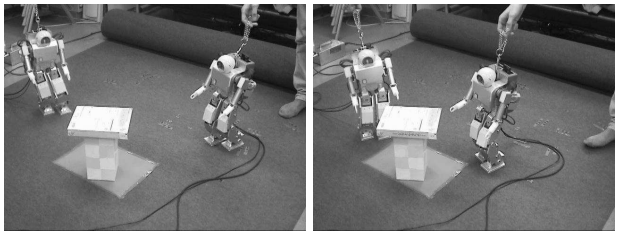


Figure 15. Two humanoid robots approach a target object in a cooperation task.

We have been applying our techniques to more complicated tasks with real-robot learning. For example, we are now trying to solve a cooperation task with a set of heterogeneous robots (see Fig.15). This is to show the effectiveness of our approach as a multi-agent learning technique. We have acquired promising performance via preliminary results. We will conduct further experiments in pursuit of the applicability of our proposed approaches.

References

[1] M. Asada, S. Noda, S. Tawaratsumida, and K. Hosoda. Purposive behavior acquisition for a real robot by vision-based reinforcement learning. *Machine Learning*, 23:279–303, 1996.

[2] R. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, January 1986.

[3] K. L. Downing. Adaptive genetic programs via reinforcement learning. In *Proc. of the Third Annual Genetic Programming Conference*, 1998.

[4] D. Floreano, S. Nolfi, and F. Mondada. Competitive coevolutionary robotics: From theory to practice. In R. Pfeifer, editor, *Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior*. Cambridge, MA, MIT Press-Bradford Books, 1998.

[5] H. Iba. Emergent cooperation for multiple agents using genetic programming. In *Parallel Problem Solving form Nature IV (PPSN96)*, pages 32–41, 1996.

[6] H. Iba. Multi-agent reinforcement learning with genetic programming. In *Proc. of the Third Annual Genetic Programming Conference*, 1998.

[7] H. Iba. Evolving multiple agents by genetic programming. In L.Langdon, W. U.-A., and P. Angeline, editors, *Genetic Programming 3*, pages pp447–466. MIT Press, 1999.

[8] S. Kamio, H. Mitsuhasi, and H. Iba. Integration of genetic programming and reinforcement learning for real robots. In *Proc. of the Genetic and Evolutionary Computation Conference (GECCO2003)*, 2003.

[9] H. Kimura, T. Yamashita, and S. Kobayashi. Reinforcement learning of walking behavior for a four-legged robot. In *40th IEEE Conference on Decision and Control*, 2001.

[10] J. R. Koza. Evolution of subsumption using genetic programming. In *Toward a Practice of Autonomous Systems*, pages pp110–119, 1992.

[11] J. R. Koza. *Genetic Programming, On the Programming of Computers by means of Natural Selection*. MIT Press, 1992.

[12] H. Liu and H. Iba. Multi-agent learning of heterogeneous robots by evolutionary subsumption. In *Proc. of the Genetic and Evolutionary Computation Conference (GECCO2003)*, 2003.

[13] OPEN-R Programming Special Interest Group. *Introduction to OPEN-R programming (in Japanese)*. Impress corporation, 2002.

[14] W. F. Punch and W. M. Rand. GP+echo+subsumption = improved problem solving. In *Genetic and Evolutionary Computation Conference (GECCO2000)*, pages pp411–418, 2000.

[15] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An introduction*. MIT Press in Cambridge, MA, 1998.

[16] Y. Takahashi, M. Asada, S. Noda, and K. Hosoda. Sensor space segmentation for mobile robot learning. In *Proceedings of ICMAS'96 Workshop on Learning, Interaction and Organizations in Multiagent Environment*, 1996.

[17] M. Terao and H. Iba. Controlling effective introns for multi-agent learning by genetic programming. In *the Genetic and Evolutionary Computation Conference (GECCO2000)*, pages pp419–426, 2000.

[18] K. Yanai and H. Iba. Multi-agent robot learning by means of genetic programming: Solving an escape problem. In *Evolvable Systems: From Biology to Hardware, 4th International Conference (ICES2001)*, pages pp192–203, 2001.

[19] C. H. Yong and R. Miikkulainen. Cooperative coevolution of multi-agent systems. Ai01-287, Department of Computer Sciences, University of Texas at Austin, 2001.