Robotica (2005) volume 23, pp. 139–147. © 2005 Cambridge University Press
doi:10.1017/S0263574704000803 Printed in the United Kingdom

Teaching robots to plan through Q-learning

Dongbing Gu and Huosheng Hu

Department of Computer Science, University of Essex, Wivenhoe Park, Colchester CO4 3SQ (UK)
E-mail: dgu@essex.ac.uk, hhu@essex.ac.uk

(Received in Final Form: June 21, 2004)

SUMMARY

This paper presents a Q-learning approach to state-based planning of behaviour-based walking robots. The learning process consists of a teaching stage and an autonomous learning stage. During the teaching stage, the robot is instructed to operate in some interesting areas of the solution space to accumulate some prior knowledge. Then, the learning is switched to the autonomous learning stage to let the robot explore the solution space based on its prior knowledge. Experiments are conducted in the RoboCup domain and results show a good performance of the proposed method.

KEYWORDS: Robot learning; Reinforcement learning; Behaviour co-ordination; State-based planning

1. INTRODUCTION

The major challenge for programming autonomous mobile robots comes from dynamic environments, uncertain sensory data, imperfect actuators and real-time requirements. To address the challenge, it is necessary to endow the robot with reactive behaviour, planning and learning abilities. Therefore, the robot-programming methodology gradually evolves from deliberative planning,[1] behaviour-based architectures,[2,3] hybrid systems,[2] to state-based planning.[4,5]

In general, the pure planning method is well appropriate to task-oriented planning which, however, heavily relies on knowledge about the real world in order to predict its actions. The main difficulty lies in dynamic or unpredictable environments since frequent re-modelling and re-planning deteriorates robot performance. The behaviour-based method allows actions to directly respond to sensory inputs with less computation. The autonomous system can be built up from simpler behaviours, and gradually integrate more complex behaviours. However, the behaviour-based robot is difficult to be scaled up to large-scale systems since it lacks mechanisms for managing complexity[6] and it is unable to use real-world models even if they are available.[7]

In contrast, the hybrid system method integrates both deliberative planning and reactive behaviours into one entity. A middle layer between deliberative planning and reactive behaviours is used for their co-ordination, which makes the control system become three-layer architectures.[8] In these hybrid systems, the responsibility of each level is not strictly defined. In most cases, the middle layer separates the planning layer from the reactive layer.[9] Since different hybrid systems are applied to different tasks, they are far from being compatible. Lack of learning capability is common to all hybrid systems due to the difficulty of integrating the learning component into the system. Up to now, many robotic systems have been built based on two-layer architectures,[10] though their co-ordination mechanisms may vary. For instance, a voting mechanism is adopted in DAMN where each behaviour votes for or against a set of actions.[11] Maes used a selection mechanism to arbitrate the behaviours.[12] Fuzzy logic is employed[13] to co-ordinate behaviours.

Recently, the state-based method has been proposed to model the dynamics of the behaviour co-ordination,[5,14–16] in which the behaviour co-ordination is defined as a Markovian Decision Process (MDP). Learning ability can be included into the MDP by using reinforcement learning.[17,18] Some typical examples include Q-learning algorithms for foraging robots,[16] an Augmented Markov Model (AMM) for interaction dynamics,[14] a state-based reinforcement learning for a legged robot to learn its locomotion gaits,[15] the Partially Observable Markov Decision Process (POMDP) to model planning,[19] and a POMDP for navigation planning.[5]

This paper focuses on the state-based planning. The objective is to develop a learning algorithm for the planning algorithm based on the two-layer architecture that can accommodate planning, reaction, and learning. Behaviours are assumed to be pre-designed, and the robot employs Q-learning to acquire its planning results. Due to the extensive learning time for real robots, a behaviour-based Q-learning is used in our algorithm instead of AMM and POMDP. Our algorithm is similar to the Q-learning algorithm in foraging robots[16] apart from a two-stage learning scheme. Although the abstraction of states and actions enables Q-learning to operate only within a limited discrete space, the learning system knows nothing about the interaction between the robot and its environment at the early stage. The learning system has to choose arbitrarily actions to try until a reward is received. For reinforcement learning, incorporating prior experience into a learning system is an efficient way. Lin used hand-coded sequences of experience to bootstrap information into value functions in order to speed up the learning process.[20] Other similar researches are carried out for individual behaviour learning.[21,22] In this paper, a teaching method is employed to bootstrap human experience into Q values, and then a two-stage Q-learning is developed. In the first stage, a human operator chooses behaviours for the robot to run. The robot accumulates the human operator's experience in the form of Q values. In the second stage, the robot treats these prior Q values as initial

settings and autonomously updates them during exploring its environment.

The rest of the paper is organised as follows: Section 2 presents how to model the state-based planning, including a brief introduction to Q-learning, the behaviour design for abstraction of the continuous motor actions, and the feature state definition for abstraction of sensory physical states. Section 3 describes the learning method for the state-based planning, including the design of heterogeneous rewards, Q-learning bootstrap, and autonomous learning. Section 4 reports some experiments and evaluation, including the experimental set-up and results. Finally, conclusions and future work are summarised in Section 5.

2. Q-LEARNING SETTING IN BEHAVIOUR BASED ROBOTICS

In behaviour-based robotics, the task-oriented planning is achieved by co-ordinating behaviours. The state-based planning is to model the co-ordination mechanism or the dynamic interaction between a robot and its environment by using a Markovian Decision Process (MDP). In a MDP, the robot observes the current state s_t, takes an action a_t, moves into next state s_{t+1}, and gains a reward r_t. Within this model, the planning objective of a robot is to find mappings from states s_t to actions a_t through maximising the accumulating rewards:

$$\sum_{t=0}^{\infty} \gamma^t r_t \tag{1}$$

where γ is a discount factor. Q-learning can be used to achieve the planning objective based on continuous interaction between a robot and its environment.

2.1. Q-learning

When a robot interacts with its environment, it can collect a data tuple (s_t, a_t, r_t) at each time step t. After a learning episode is completed, a set of data tuple is collected. In the discrete Q-learning, both states s_t and actions a_t are discrete. The collected data is compressed into a table $Q(s_t, a_t)$ that is an estimated accumulated rewards when the robot is in the state s_t and takes the action a_t. The Q table can be updated during learning when more episodes are run. In the one-step Q-learning algorithm, Q values of all state and action pairs are updated by the following learning rule:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_t + \gamma \cdot \max_a Q(s_{t+1}, a) - Q(s_t, a_t)) \tag{2}$$

After learning, the final optimised action a_t^* at the state s_t can be acquired by:

$$a_t^*(s_t) = \arg\max_{a_t} Q(s_t, a_t) \tag{3}$$

In a Markovian environment where the next state s_{t+1} only depends on the current state s_t and the action a_t and is independent to any previous states, Watkins and Dayan have proved that Q-learning is convergent after the visit of each state-action pair is infinitely often.[23] But it is impractical for real robots to achieve this during learning and so is for simulation. Nevertheless, no matter how complex about the convergence, there have been a lot of applications of reinforcement learning so far since the learning results have a potential to provide a good solution even not an optimal one.

Real robots have continuous sensory physical states and continuous motor actions. For the continuous Q-learning, the learning becomes more difficult since the learning algorithm cannot explore entire continuous solution space. However, the learned Q values can be generalised to the entire solution space through either a neural network for the continuous state space,[24] or some heuristic rules for a continuous action space.[25] In this paper, our method does not adopt the generalised approach to handle the continuous spaces. Instead, behaviours and feature states are defined as the actions and states in the discrete Q-learning algorithm. The continuous motor actions are abstracted as temporally extended actions – behaviours and the continuous sensory physical states are abstracted as physical events – feature states. Then, the discrete Q-learning algorithm is used to implement the learning.

The extracted feature states and behaviours are the abstraction of sensory states and motor actions. They can retain the major physical state and action features and ignore less useful details. The abstraction separates entire sensory state space into a number of connected local regions where local low-level control strategies are easily achieved. The high-level control strategies are left for planning that is responsible to the transition between these local regions.

2.2. Robot behaviours

For a specific task, a group of primary behaviours can be developed according to domain knowledge about the task. These behaviours are the basic skills that the robot should have in order to accomplish the task. In associated with the local regions of the state space, each behaviour operates within one of them until the perceived states have been changed. Therefore, a behaviour can be viewed as a temporally extended action. When a robot operates within the regions, the corresponding behaviours maps current sensory states into motor actions and handles uncertainties. The limited ranges of local regions ease the design of behaviours that are only required to handle uncertainties in local regions, rather than in global regions.

In this research, Fuzzy logic controllers (FLC) and decision trees are used to design behaviours. In a FLC, uncertainty is represented by fuzzy sets and an action is generated co-operatively by several rules, each one triggering to some degree to produce smooth, reasonable and robust control effects. The design of a FLC can be based on human experience.[26] The behaviours are designed independently and keep unchanged during the learning discussed in this paper. The following five behaviours are pre-designed for Sony legged robots to play soccer.

- *Find-ball behaviour (FB)*: The robot starts to scan the pitch through its onboard camera. Once the ball is found, the

https://doi.org/10.1017/S0263574704000803 Published online by Cambridge University Press

robot updates the ball's position and its confident value. Simultaneously the robot turns its heading toward the ball.

- *Chase-ball behaviour (CB)*: The robot moves its head to track the ball, and moves towards the ball. The sensory states of this behaviour include ball position, goal position, and robot position.
- *Align with goal behaviour (AG)*: The robot slightly adjusts its position to align the ball, its orientation with the goal in order to kick the ball into the goal. The sensory states of this behaviour include ball position, goal position, and robot position.
- *Kick-ball behaviour (KB)*: some specialised kick skills are applied.
- *Dribble-ball behaviour (DB)*: The robot kicks the ball by using its feet and keeps its body moving. It's very useful when the ball is close to edges or corners.

2.3. Feature states

Based on the behaviour design, task-oriented planning operates within the cognition level and reflects high-level mental states. The details of sensory data may not contribute too much for planning, and may instead trap it into deadlocks or local optima. For the learning stage, the algorithm that directly searches on sensory data could meet an inhibited large state space. It is necessary to extract the state features in sensory data to ignore information that is less useful in high-level reasoning. Often, for robot tasks, some significant physical events are existed, and represent critical points in a sensory state space and can be described by binary values. The combination of the significant physical events forms a feature state space. Planning can be conducted within the feature state space.[15,16]

Formally, each physical event can be represented by a binary state p ($p = 1$ for the event occurring). For a specific task, there are n physical events $(p_1, \ldots, p_n)$ that can be defined to characterise the task. The combination of these events or binary values constitutes a feature state $s = p_1 \ldots p_n$. There are 2^n feature states in total in the feature state space S. Decision trees or clustering techniques can be used to map sensory physical states into feature states. In this research, four physical events are defined for a legged robot to play soccer.

- *Is the ball found (p_1)*: simply compare the ball confidence value with a threshold.
- *Is the ball near enough (p_2)*: since the robot always tracks the ball, its tilt angle and pan angle show if the ball is just in the front of the robot.
- *Is the robot behind the ball (p_3)*: it depends on the distance and angle between the ball and the robot, and the angle between the robot and the goal.
- *Is the robot aligned with the goal (p_4)*: it again depends on the information same as p_3, but with more accurate values.

For example, $s = 1010$ represents the robot find the ball and is behind the ball, but not near enough and not align with the goal.

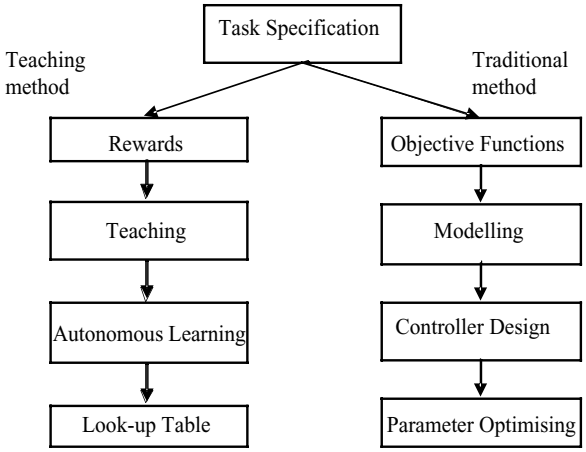

Fig. 1. A comparison between the teaching method and traditional optimising design method.

3. LEARNING SCHEME

Using reinforcement learning, and Q-learning in particular, to acquire control algorithms for robots enables human designers to concentrate on task description in the form of rewards. Normally, designing rewards is generally easier than designing control algorithms, especially in the noisy and unpredictable environment. The control algorithms are acquired automatically through learning guided by rewards. Figure 1 shows both the teaching method (left branch) and the traditional design method (right branch). Given task specifications, the objective of both methods is to acquire final control algorithms. More specifically,

- In the traditional design method, objective functions are first established according to task specifications. It is followed by modelling control environments and designing controller structures. Finally, controller parameters are optimised through optimisation. For real robots, modelling is a difficult step due to the uncertainty and unpredictability in the real world.
- In the teaching method, task specifications are first interpreted into rewards. It is followed by leading the robot to some interesting areas through teaching. Based on the learned knowledge about interesting areas, the robot autonomously explores its environment to update utility values of state-action pairs. Finally, the robot simply searches the look-up table to find those state-action pairs with maximum utility values.

3.1. Heterogeneous rewards

In Q-learning, the learner executes a policy, accumulates and updates the estimated rewards. The rewards can be singular values to indicate goals or sub-goals and usually cannot be gained immediately. For some tasks, rewards are designed to show if the goal is achieved. These rewards are sparse in both temporal and spatial senses. For complex tasks, Q-learning takes a long time when only sparse rewards are available. For real robots, dense rewards[22] or progress estimators[16] are necessary.

The rewards are the payoff returned from the environment to indicate the effect of robot's actions taken so far. Due to

https://doi.org/10.1017/S0263574704000803 Published online by Cambridge University Press

noisy sensory data, the dense rewards may contain uncertainty as well or be wrongly interpreted by perceptual algorithms. The aliasing perceptual is more possibly occurred in a high level of abstraction. Therefore, it is necessary to provide the accurate information about the interaction between robots and environments to the learned robots. The information should be produced from independent sources that act as assessors to evaluate robots behaviours that learned, instead from onboard sensors that are used to perceive environment states.

In this research, a monitor system is adopted as an assessor. The robot uses both sparse and dense rewards during its learning and these rewards come both from on-board sensors and the monitor system. These heterogeneous rewards include:

- Task reward: this is the most significant reward for a learning robot. It is rewarded for their achievement toward the goal. For soccer playing robots, shooting a goal will be rewarded by the monitor system.
- Sub-task rewards: these rewards are provided to evaluate the performance of individual behaviours. They can be sub-task goals denoted by sparse values or progress estimation in a dense form. Two rewards are used in the soccer playing robots. One is the distance between the robot and the ball, which is provided by the monitor system. Another is the angle distance between a robot and the ball, which is taken from on-board sensors. These rewards can continuously evaluate the robot performance.

3.2. Teaching stage

The teaching stage is to provide initial Q values to Q-leaning algorithms in the second stage in order to guide learning to search within interesting areas rather than blindly explore solution spaces at the early learning stage. With these prior experiences, the robot could behaviour reasonably.

The teaching consists of several lessons, each of which starts from the robot's initial position and runs until the terminal condition is met. The condition is either scoring a goal or over time. Figure 2 shows the flowchart of the teaching processing. There are two loops that run in parallel on both the robot and the computer in the monitor system. Two loops run asynchronously and communicate with each other through the Internet. When the robot is started, it localises itself first in order to extract environment information. The feature states are then sent to the monitor system to help the teacher to select behaviours. The robot executes one step of the behaviour selected by the teacher. Then it checks if the terminal condition is met. If not, the robot goes to send feature states and run the inner loop again. Finally the robot stops moving and starts to run the off-line Q-learning algorithm to update the Q table that is sent to the monitor system. The robot completes its current lesson and goes back to localise again for next lesson.

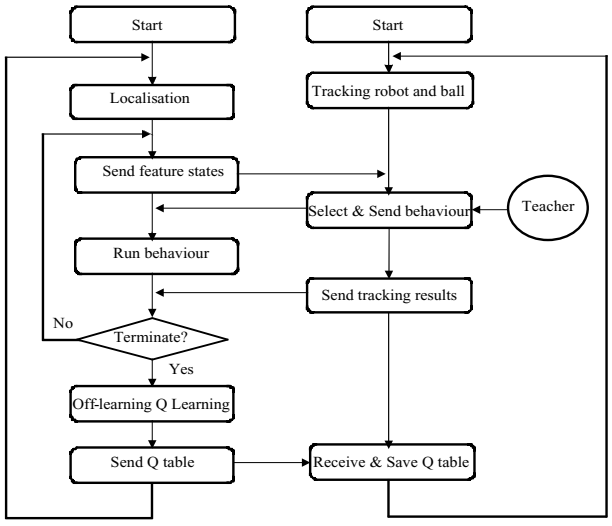

Fig. 2. The flowchart of the teaching processing.

The monitor loop starts from the object tracking algorithm that searches images and provides the ball and robot's position. The teacher (human operator) selects one of behaviours according to his/her view of the field and feature states received from the robot. The selected behaviour and tracking results are both sent to the robot for its learning process. The monitor system also records all of the learnt results, i.e. Q tables, when one lesson is completed.

During the teaching process, the Q table is the knowledge learned from lessons. The robot completely follows instructions (the behaviours) and collects whole data set of the behaviour, feature states, and rewards. After one lesson, it employs the off-line Q-learning algorithm to update the Q table. Although the whole data set is also available to the monitor computer, learning is still executed on the robot's side since data in the monitor system is delayed by its communication channel.

The teacher plays a key role in the teaching process, and decides which behaviour is to run next. The information available is from two views of the environment: the teacher's own view of the field and the robot's view (feature states). Since a behaviour is a temporal extended action (not a single action), the teacher has time to make decisions. During the run of a behaviour, the robot has to decide which low-level action to execute based on sensory data and behaviour's rules. The teacher may have mistakes (the wrong behaviour is selected), which means that the lesson may contain wrong data. However, the knowledge obtained in the teaching is only used for initial values in the next Q-learning stage. The final result of the teaching is a Q table that represents prior experiences or part solutions.

3.3. Autonomous learning stage

In this second stage learning, the robot starts to learn autonomously without instructions from a teacher. It is a standard Q-learning process, in which the robot needs to explore the solution space and exploit the learned experience. The monitor system is still required to provide rewards and to save the Q table, but without the need for instructions. The learning system still follows the similar procedure as shown in Figure 3. Only three different steps are taken.

- The robot has to select its behaviour rather than follow the received one. The ε-greedy method is employed to balance exploration and exploitation. With the prior experience learned in the teaching stage, the robot can act reasonably

https://doi.org/10.1017/S0263574704000803 Published online by Cambridge University Press

1) *Initialisation. The initial behaviour is set as Find-ball behaviour.*
2) *If enough learning episodes are completed, exit*
3) *Self-localisation.*
4) *Loop.*
   a. *Perceive environment.*
   b. *Extract the feature states.*
   c. *If the feature states have changed*
      i. *Calculating the rewards.*
      ii. *Q value updating using (2)(3).*
      iii. *Select a behaviour using the ε-greedy algorithm.*
   d. *Run the behaviour.*
   e. *If the terminal conditions are met*
      i. *Send Q table back to the monitor system*
      ii. *Back to step 2 for next episode.*
   f. *Otherwise, back to step 4) for next loop.*

Fig. 3. The on-line Q-learning algorithm.

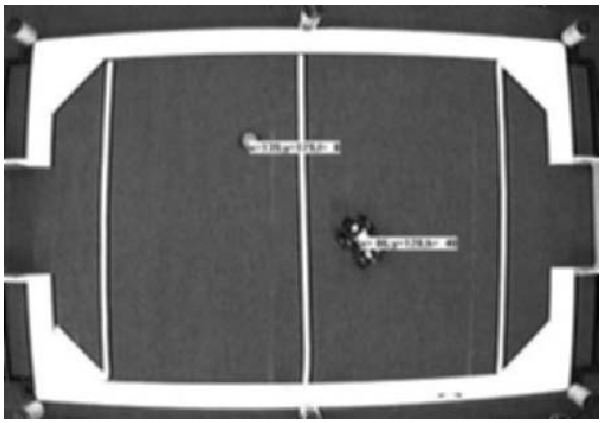

Fig. 4. The top view of the playing field from the overhead camera.

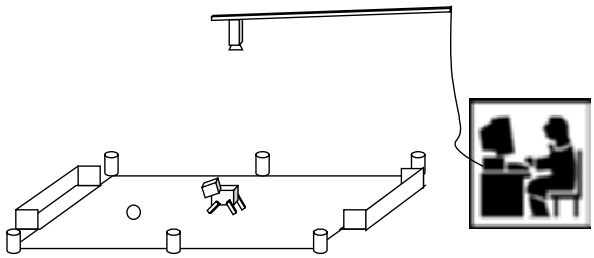

Fig. 5. The monitor systems and the operator.

rather than randomly though some behaviours are selected randomly with a small exploration probability.

- An on-line Q-learning version is implemented to replace off-line Q-learning in Figure 3. It means that Q values are updated when the robot changes from one feature state to another while no need to wait for the terminal condition.
- There is no teacher or human operator to intervene the learning process at the monitor system that works as a server to supply the rewards to the robot client.

The temporal credit assignment problem is solved by TD(λ), in which a replacing eligibility trace e(s, a) is used to distribute rewards, and records the occurrence of the visiting of state-action pair (s, a). The formula (2) can be rewritten as follows.

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_t + \gamma \cdot \max_a Q(s_{t+1}, a) - Q(s_t, a_t))e(s_t, a_t) \quad (4)$$

$$e(s, a) = \begin{cases} 1 & s_t = s, a_t = a \\ e(s, a)\gamma\lambda & otherwise \end{cases} \quad (5)$$

where s and a represent the current behaviour and the feature state respectively.

The on-line Q-learning algorithm is as follows. The Q value is updated every loop in Step 4) only when feature states changes (at step c).

4. EXPERIMENTS

4.1. The robot and its monitor system

Sony legged robots are quadruped robots that resemble the basic behaviour of dogs. They are controlled by an embedded R4000 microprocessor with over 100 MIPS performance. There are 20 motors for action. The main sensors include 20 encoders for motion control of 20 motors, a colour CCD camera, an infrared range sensor, 3 gyros for posture measurement (roll, pitch, yaw), and touch sensors. Additionally, there is a stereo microphone and a loud speaker for communication.[27]

The playing field for the Sony Legged Robot League is 4 m in length and 3 m in width. Figure 4 shows a top view of the playing field from the overhead camera. The goals are centred on both ends of the field, and are 60 cm wide and 30 cm high. Six unique coloured landmarks are placed around edges of the field, with one at each corner and one on each side of the halfway line. Each landmark is painted with two different colours of which pink is either at the top of landmarks on the one-side of the pitch or at the bottom of landmarks on the other side. These landmarks are used by robots to localise themselves within the field. The ball, walls, goals, landmarks and robot uniforms are painted with eight different colours distributed in the colour space so that a robot can easily distinguish them.

A global monitor system is set up in our laboratory as shown in Figure 5, which includes an overhead camera, a desktop computer, and visual tracking software.[28,29] The monitor recognises the robot and the ball according to their colours. Through image processing, the monitor updates their positions continuously. The tracking results are displayed via images (see values in Figure 4). These tracking results and judgement about scoring are transferred to the robot. The robot and the monitor computer are connected through the wireless network with client/server architecture. The robot works as a client and continuously asks for information from the server. The monitor system updates its image and tracking results in the rate of about 30 frames per second and the robot operates in the rate of about 20 frames per second, which is adequate to the learning process being conducted.

4.2. Results

All experiments are conducted in five situations, each of which is called a play and has different setting for initial ball

https://doi.org/10.1017/S0263574704000803 Published online by Cambridge University Press

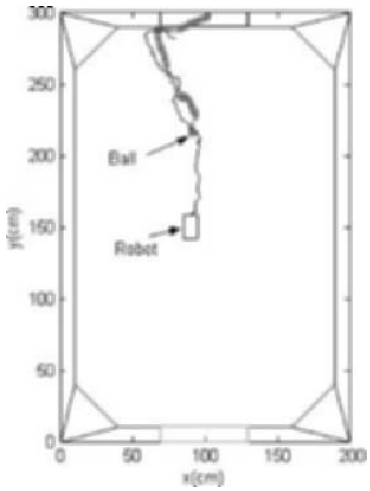

Fig. 6. Play 1.

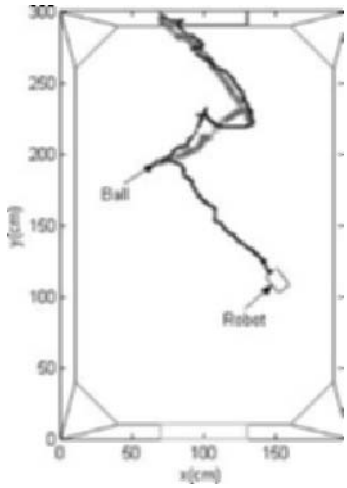

Fig. 7. Play 2.

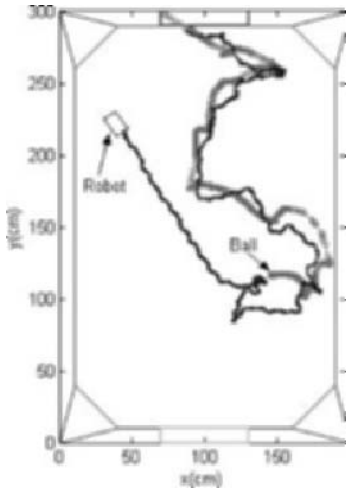

Fig. 8. Play 3.

position and initial robot position. Figure 6 to Figure 10 show these five plays with the arrows pointing to initial positions. The parameters of Q-learning algorithms are chosen as:

$$\alpha = 0.1, \quad \varepsilon = 1\%, \quad \lambda = 0.9, \quad \gamma = 0.9$$

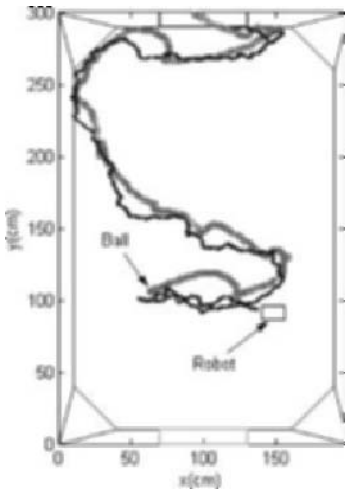

Fig. 9. Play 4.

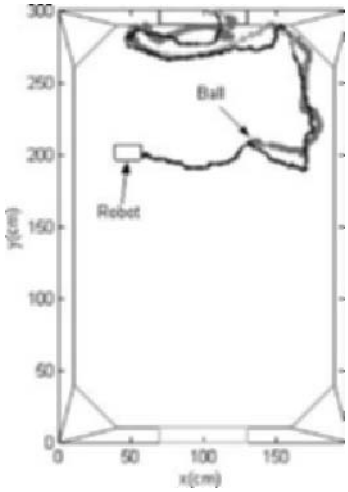

Fig. 10. Play 5.

The Q-learning terminal conditions are either that a goal is scored or that 2000 steps are executed. The experiment procedure is as follows:

Phase 1. Teaching stage: 5 teaching lessons are taken. For each play, one lesson is run. The off-line Q-learning is preceded at the end of each lesson.

Phase 2. Demonstration after teaching: 30 trials are run after teaching (the off-line Q-learning) by using those behaviours with maximum Q values. For each play, 6 trials are executed.

Phase 3. Autonomous learning stage: 30 episodes are run by using the on-line Q-learning algorithm shown in Figure 3. For each play, 6 episodes are implemented.

Phase 4. Demonstration after autonomous learning: 30 trials are run after on-line Q-learning by using behaviours with maximum Q values. For each play, 6 trials are implemented.

Phase 2 is an off-policy learning process where the executed behaviours are selected by a teacher, which are not necessarily same as behaviours that the robot wants to learn.

https://doi.org/10.1017/S0263574704000803 Published online by Cambridge University Press

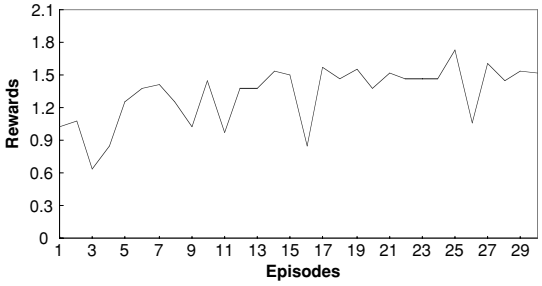

Fig. 11. The average received rewards.

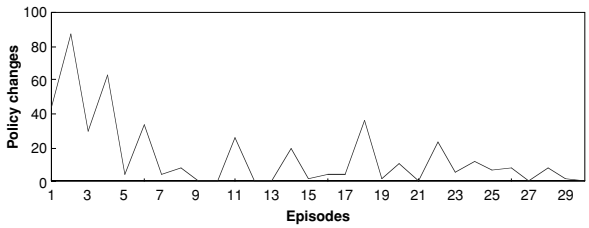

Fig. 12. The number of the policy changes.

Phase 4 is an on-policy learning process where the behaviours executed are determined by the ε-greedy method and only 1% behaviours are selected randomly. Figure 11 shows the increasing trend of the average received rewards received during Phase 4. Several decreasing points in the curve reflect the exploration mechanism. The number of policy changes also indicates that learning is going toward steady results, as shown in Figure 12.

The estimated reward is distributed into a Q table. Figure 13 shows the change of Q values in two feature states during the learning process. The first 5 episodes are run for Phase 1 teaching and other 30 episodes are run for on-line Q-learning. The CB behaviour won at the end of teaching in both feature states. Therefore the CB behaviour is used for these two feature states after the teaching stage. However, the situation changes during on-line Q-learning where the DB behaviour gradually outperforms the CB behaviour as shown in Figure 13(a) and the FB behaviour outperforms the CB behaviour in Figure 13(b). The function of the teaching is to find some useful behaviours in the feature states in order to bootstrap the on-line Q-learning to save the learning time. The decreasing of the maximum Q value in Figure 13(a) demonstrates that the knowledge acquired during the teaching can be corrected by Q-learning.

At the autonomous learning stage, 30 learning trials are run in six rounds with the order play1, play2, play3, play4, and play5. The average steps in each round are calculated and displayed in Figure 14. It indicates that both average steps in six rounds and their standard derivations are decreased. The robot gradually scores a goal in less steps or shorter time.

The evaluation of experiments are further carried out in terms of average steps of a trial, average rewards of a trial, and the number of failures for scoring a goal in 2000 steps. In Figure 15, average steps of trials for both Phase 2 and Phase 4 are compared. The white bars represent the results after teaching (phase 2) and the grey bars represent the results

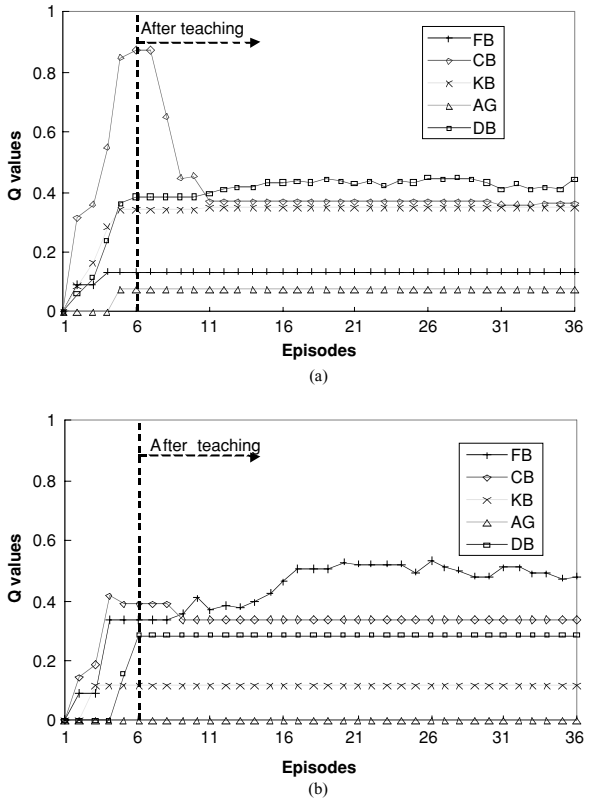

Fig. 13. The Q value changes in two feature states during the Q-learning.

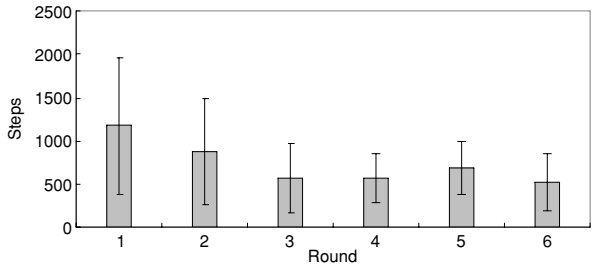

Fig. 14. The average steps of a trial in six rounds.

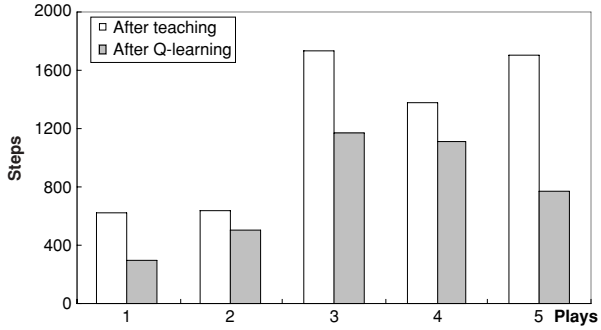

Fig. 15. The average steps of a trial in a play.

after autonomous learning within Phase 4. Due to different situations in five plays, they are separately compared.

In all five plays, the average step used to score a goal after autonomous learning is less than those used just after teaching at Phase 1. The grey bars also show that average

https://doi.org/10.1017/S0263574704000803 Published online by Cambridge University Press

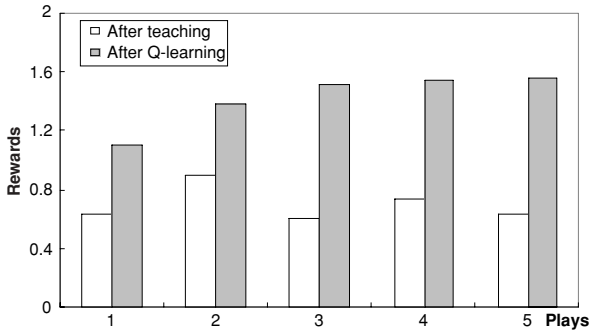

Fig. 16. The average rewards of a trial in a play.

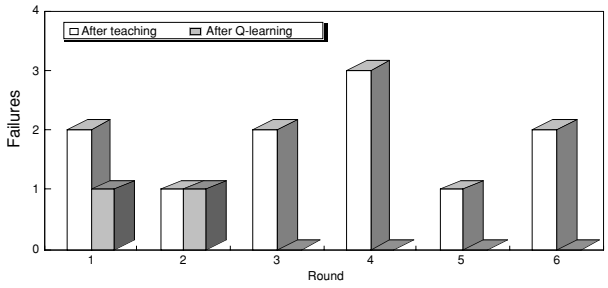

Fig. 17. The total number of failures after 1500 steps for each round.

steps used in five plays increase in the order of play1, play2, play5, play4, and play3. This order coincides with difficulties in initial settings shown in Figure 6 to Figure 10. Figure 16 shows average received rewards of a trial. Again in each play, the results after autonomous learning are better than those just after teaching at Phase 1.

The robot may fail to score a goal during a trial due to the uncertain nature of experiments. Figure 17 shows the total number of failures after 1500 steps for each round. The number in each five plays after autonomous learning is less or equal to the number after teaching. The five successful plays are displayed in Figures 6 to 10. The dark curves represent the robot's trajectories and the light curves represent the ball's trajectories. The robot can move to the ball and manoeuvre the ball into the goal. It demonstrates that the robot acquires the state planning strategy after learning.

5. CONCLUSIONS AND FUTURE WORK

This paper describes the application of Q-learning algorithms to the mapping from sensory physical states into motor actions. The challenges include the large continuous solution space and uncertainties in real robots. Some measures are taken to address these challenges. Firstly, continuous spaces are converted into discrete spaces. It is implemented by the behaviours design for motor actions and the feature states extraction for sensory physical states. Secondly, two-stage learning is developed to speed up the learning process for real robots. After the robot gains prior experiences through teacher's instructions at the first stage, the robot uses these experiences to explore interesting areas in the solution space rather than randomly searching. Finally the heterogeneous rewards are employed to guide the robot towards optimal points. They can be provided from both a global monitor system and robot's on-board sensors. All efforts are made to permit the robot to speed up the exploration process.

The approach developed in this paper can also be viewed as high-level planning. Although the combination of reactive behaviours and task-oriented planning has been a challenge for many robotic systems, the learning method discussed in this paper makes use of both state-based planning and Q-learning to implement the control design. Based on this method, the planning tasks have been implemented by designing high-level rewards, and make the robot be able to acquire low-level control algorithms through Q-learning.

Although the behaviours and the feature states discussed in this paper are all pre-defined, both of them can be embedded into the learning system to be further modified through learning. This can be achieved by keeping the high-level planning unchanged and evolving low-level rules, which is to be investigated.

References

1. J. Albus, "Outline for a Theory of Intelligence," *IEEE Transactions on SMC* **21**(3), 473–509 (1991).
2. R. C. Arkin, *Behaviour-based Robotics* (The MIT Press, 1998).
3. R. Brooks, "A Robust Layered Control System for a Mobile Robot," *IEEE J. Robotics and Automation* **2**(1), 14–23 (1986).
4. J. Kosecka and R. Bajcsy, "Discrete Event Systems for Autonomous Mobile Agents," *Robotics and Autonomous Systems* **12**, 187–198 (1994).
5. L. D. Pyeatt, "Integrating of Partially Observable Markov Decision Processes and Reinforcement Learning for Simulated Robot Navigation," *PhD Thesis* (Colorado State University, 1999).
6. R. Hartley and F. Pipitone, "Experiments with the Subsumption Architecture," *Proc. of IEEE International Conference on Robotics and Automation* (April 9–11, 1991) pp. 1652–1658.
7. D. Kirsh, "Today the Earwig, Tomorrow Man," *Artificial Intelligence* **47**, 161–184 (1991).
8. E. Gat, "On Three-Layer Architecture," **In**: *Artificial Intelligence and Mobile Robots* (D. Kortenkamp, R. P. Bonnasso and R. Murphy, eds.) (MIT/AAAI Press, 1998) pp. 195–210.
9. R. Volpe et al., "The CLARAty Architecture for Robotic Autonomy." *Proc. of the IEEE Aerospace Conference* (Big Sky Montana, 2001) pp. 10–17.
10. P. Pirjanian, "Behaviour Co-ordination Mechanism – State-of-the -art," *Tech-report IRIS-99-375* (Institute for Robotics and Intelligent Systems, School of Engineering, University of Southern California, 1999).
11. J. K. Rosenblatt, "DAMN: A Distributed Architecture for Mobile Navigation," *PhD Thesis* (Carnegie Mellon University, 1997).
12. P. Maes, "Situate Agents Can Have Goals," *Robotics and Autonomous Systems* **6**, 49–76 (1990).
13. A. Saffiotti, E. Ruspini and K. Konolige, "Using Fuzzy Logic for Mobile Robot Control," **In**: *Practical Applications of Fuzzy Technologies* (H. J. Zimmermann, Ed.) (Kluwer Academic, 1999) pp. 185–206.
14. D. Goldberg, "Evaluation the Dynamics of Agent-Environment Interaction," *PhD Thesis* (University of Southern California, 2001).
15. M. Huber, "A hybrid Architecture for Hierarchical Reinforcement Learning," *Proc. of IEEE International Conference on Robotics and Automation* (Detroit San Francisco, 2000) pp. 3290–3295.
16. M. J. Mataric, "Reinforcement Learning in the Multi-Robot Domain," *International Journal of Autonomous Robots* **4**(1), 73–83 (1997).

https://doi.org/10.1017/S0263574704000803 Published online by Cambridge University Press

17. R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning* (MIT Press, 1998).
18. L. P. Kaelbling and A. W. Moor, "Reinforcement Learning: A Survey," *Journal of Artificial Intelligence Research* **4**, 237–285 (1996).
19. S. Koenig and R. G. Simmons, "Unsupervised Learning of Probabilistic Models for Robot Navigation," *Proc. of the 1996 IEEE International Conference on Robotics and Automation* (1996) pp. 2301–2308.
20. L. J. Lin, "Self-improving Reactive Agents Based on Reinforcement learning, planning, and teaching," *Machine Learning* **55**, 293–321 (1992).
21. M. Dorigo and M. Colombetti, "The Role of the Trainer in Reinforcement Learning," *Proc. of MLC-COLT '94 Workshop on Robot Learning* (July 10th, New Brunswick, NJ, USA, 1994) pp. 37–45.
22. W. D. Smart and L. P. Kaelbling, "Effective Reinforcement Learning for Mobile Robots," *Proc. of the IEEE International Conference on Robotics and Automation* (2002) pp. 3404–3410.
23. C. J. Watkins and P. Dayan, "Technical note: Q-learning," *Machine Learning* **8**(3/4), 323–339 (1992).
24. G. A. Rummery, "On-Line Q-Learning Using Connectionist Systems," *Technical Report CUED/F-INFENG/TR 166* (Cambridge University, 1994).
25. J. R. Millan, D. Posenato and E. Dedieu, "Continuous-Action Q-Learning," *Machine Learning* **49**, 247–265 (2002).
26. D. Gu and H. Hu, "Evolving Fuzzy Logic Controllers for Sony Legged Robots," *Proc. of the RoboCup 2001 International Symposium* (Seattle, Washington, 2001) pp. 4–10.
27. M. Fujita, "Development of an Autonomous Quadruped Robot for Robot Entertainment," *Autonomous Robots* **7**, 7–20 (1998).
28. D. Golubovic and H. Hu, "An Interactive Software Environment for Gait Generation and Control Design of Sony Legged Robots," *Proc. of the 6th Int. Symposium on RoboCup* (Fukuoka, Japan, 2002) pp. 24–25.
29. B. Li, H. Hu and L. Spacek, "A Hybrid Experimental Platform for Sony Legged Robots," *Proc. of CACSCUK'2002* (UMIST, England, ISBN 0 9533890 5 9, 2002) pp. 7–12.

https://doi.org/10.1017/S0263574704000803 Published online by Cambridge University Press