6

Reinforcement Learning for Autonomous Robotic Fish

Jindong Liu, Lynne E. Parker, and Raj Madhavan

Department of Computer Science, University of Essex,
Wivenhoe Park, Colchester, United Kingdom,
`(jliua, dgu, hhu)@essex.ac.uk`

The chapter discusses applications of reinforcement learning in an autonomous robotic fish, called Aifi. A three-layer architecture is developed to control it. The bottom layer consists of several primary swim patterns. A sample-based policy gradient learning algorithm is used in this bottom layer to evolve swim patterns. The middle layer consists of a group of behaviours which are designed for specific tasks. The top layer is a Markov Decision Process (MDP), which is used for the planning purpose. The behaviour coordination is conducted by building a MDP in the top layer. A state-based reinforcement learning algorithm, Q-learning in particular, is applied in the top layer to find an optimal planning policy for a specific task. Both simulated and real experiments show good feasibility and performance of the proposed learning algorithms.

6.1 Introduction

The Human Centered Robotics Research Group at Essex has developed a number of robotic fishes since April 2003. Different from other robotic fish projects, we focus on realizing multiform fish-like behaviours and machine intelligence on our robotic fish. The aim of our research project is to make the robotic fish, named Aifi, "grow" from "baby", which is able to learn the best control parameters for its variant behaviours and learn to adapt itself to changes in its environment, such as variable water current and moving obstacles.

To achieve goal-oriented tasks and fast response ability to the dynamics in environments, Aifi is controlled based on a three-layer hybrid architecture (see Figure 6.1). From bottom to top, it comprises a *swim pattern* layer, a *behaviour* layer and a *cognitive* layer. The *swim pattern* layer classifies the swimming motion of robotic fish into several basic swimming elements, called *Swim Patterns*, which interpret the commands from the *behaviour layer* into the low level motion control. It consists of *cruise-in-straight*, *sharp-turning*, *cruise-in-turning* and *ascent/descent*. The *behaviour* layer is designed to quickly response to the sensor data and direct Aifi to apply one of swim patterns. It includes several individual behaviours: *obstacle-avoiding*, *wall-following*, *goal-seeking*, *keep-level*, *wandering*, etc. The *cognitive* layer extracts robotic fish status from the sensor data and conducts task-oriented reasoning and planning. In the *cognitive* layer, it changes the coordination parameters which are used to coordinate all individual behaviours in the *behaviour* layer.

Within this layered architecture, machine learning can be conducted separately at each layer. In the *swim pattern* layer, each swim pattern actually is represented by a series of kinematic functions of motors, which are embedded in the tail joints. The learning is designed to adjust the parameters of kinematic functions to achieve improved performance. In the *behaviour* layer, behaviours are optimized according to the encoding method of behaviours. For example, if behaviours are encoded by fuzzy logic controllers, the learning algorithm will be applied to fuzzy rules and fuzzy function parameters. In the *cognitive* layer, learning algorithms are designed to update the parameters of reasoning and planning. In this chapter, we consider the learning algorithms for the *swim pattern* layer and the *cognitive* layer.

Due to the immature fish swimming mechanism and the variety of robotic fish mechanical structures, it is much difficult to build proper models for swim patterns. As a result, most of the control parameters of robotic fish swim patterns are tuned manually, which rely on the good human expertise. This manual tuning process is normally time-consuming if an optimum solution or sub-optimum solution in practice is concerned. Additionally, the parameters tuned in such a way can only be adapted to a static environment and could not perform well if the environment changes.

Alternatively, various model-free machine learning techniques have been adopted in many bio-mimetic robot projects to find the optimized control parameters, such as biped robots [11] and Sony Aibo robots [4]. Purely policy gradient reinforcement learning was originally proposed in REINFORCE [10] where policy gradient descent was used to update policy parameters. It was further extended to include value iteration in [2] by defining errors as payoffs. In [8], authors proposed a reinforcement learning algorithm without estimating a value function. In these implementations, the policy updating is converted to the parameter updating by making the policy parameterized by a set of parameters. As it is convenient to use experiment samples to find the gradient of the learned policy with respect to parameters, sample based policy gradient reinforcement learning was successfully applied in achieving fast locomotion for the Sony dog's gaits in [5] and an autonomous robot navigation controller in [3].

In this chapter, we adopt two kinds of *Reinforcement Learning(RL)* algorithms as the basic self-learning methods for Aifi. One is a sample-based policy gradient learning, which is used to optimize the control parameters in the swim pattern layer. Another is a state-based RL, which is used to find a mapping between discrete states and actions in the cognitive layer.

The rest of this chapter is organized as follows. Section 6.2 gives a brief description of Aifi. Section 6.3 presents the implementation of the policy reinforcement learning of swim pattern control parameters. Section 6.4 addresses a typical state-based reinforcement learning in the cognitive layer. In Section 6.5, some simulated experiments and real tests are given to show the feasibility and performance of our method. Summary and future work are given in Section 6.6.

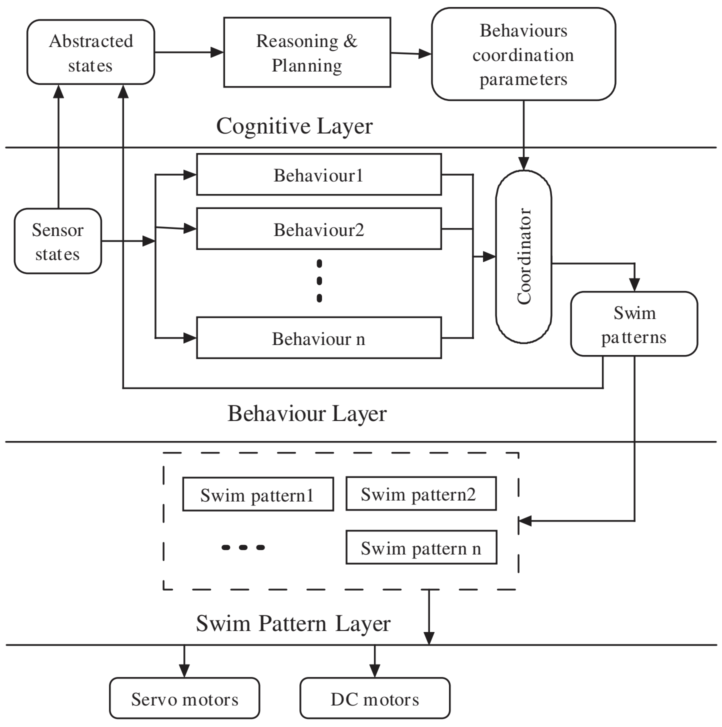

Fig. 6.1. Control structure of Aifi

6.2 Introduction of Robotic Fish-Aifi

Aifi is about 50 cm in length, 20 cm in height and 12 cm in width. It has three joints in its tail which is controlled by three servo motors. Additionally, one DC motor controls its center of gravity position and one mini-pump manages its buoyancy. The center processing unit is a cutting-edge micro-computer, Gumstix [1] which is responsible for all autonomous control computations. Aifi is equipped with several kinds of sensors to response to the dynamical changes in its environment, its position in the tank, the robot attitude and the internal status (e.g. the battery voltage). A standard configuration of Aifi includes four infrared sensors, one dual-axis accelerometer/inclinometer, one piezoelectric vibrating gyroscope, one water pressure sensor, three electric current sensors and three servo turning angle sensors. It is able to sense obstacles around it within a range of 40cm and its depth in the tank. It also can perceive the pitch/roll angle, the one-order derivative of the yaw angle, the turning angle of the tail joints and the power consumption on them. However, Aifi has no ability to localize itself in the horizontal plane because it has no sensor to measure its linear speed, thus it can not localize itself by the way of odometer used in the common mobile robots. Figure 6.2 presents the profile of Aifi used in this research.

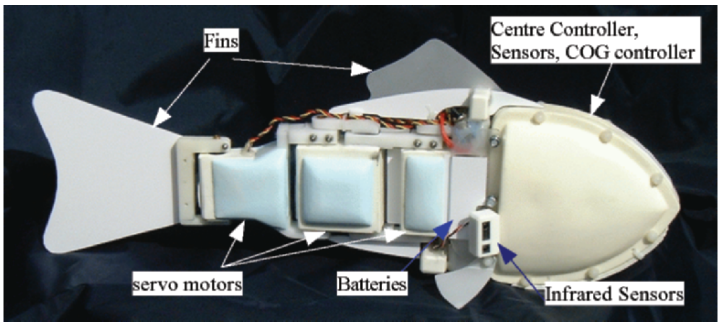

Fig. 6.2. Robotic fish-Aifi profile

6.3 Policy Gradient Learning in Swim Pattern Layer

For robotic fish applications, the advantage of using policy gradient reinforcement learning is that it can integrate the prior knowledge with later autonomous learned experience. The prior knowledge can eliminate the unreasonable parameter selection and limit the learning trace in a narrow feasible parameter space. For example, the largest turning amplitude of each joint is limited by their mechanical design. The maximum or minimum turning speed is both prior decided by the motor type and biological observation on real fishes. All of these knowledge belongs to prior knowledge. The more kinds of prior knowledge we have, the closer the initial value is to optimal. In summary, the prior knowledge is applied to set initial values and the scopes of learning parameters. This integration can shorten the learning time. For instance, the control parameters of a robotic fish can be firstly tuned manually based on any prior experience that is available. After the manual tuning, the fine tuning can be implemented by using a policy gradient reinforcement learning algorithm. Assume that the policy is differentiable with respect to each parameter, the autonomous learning is started from the manual tuning parameters. It estimates the policy's gradients in the parameter space and then updates the parameters to coverage to a local optimum.

First, we define the learning objective for each of robotic fish swim patterns according to their functions. In the policy gradient reinforcement learning, the objective is viewed as the payoff from the environment or the score of the policy function. For example, the maximum turning angle is defined as the objective of *sharp-turning* swim pattern. The payoff indicates how much benefit an agent, i.e. a robotic fish, can receive from its environment after it applies one policy. Normally, the payoff can be measured by sensors that are either on-line or off-line. In our situation, the linear speed is measured by an overhead camera; the angular speed and the power consumption are measured by an embedded compass and an electric current sensor respectively.

A policy π is defined as a probability distribution which is parameterized by the parameters extracted from the control parameters of robotic fish swim patterns. We denote these parameters as $\Theta = \left\{\theta^1, \ldots, \theta^N\right\}$. The discounted infinite payoff for this policy is defined as follows:

$$V(\pi) = \sum_{t=0}^{\infty} \gamma^t E[r_t] \tag{6.1}$$

where $\gamma (0 < \gamma < 1)$ is a discount factor, r_t is a payoff and $E[r_t]$ is the expectation of r_t.

Once we obtain an estimate of the discounted infinite payoff gradient with respect to the policy parameters $\frac{\partial V(\pi)}{\partial \theta^i}$, then the policy parameters can be updated by using the following equation:

$$\theta_{t+1}^i = \theta_t^i + \alpha_t \frac{\partial V(\pi)}{\partial \theta^i} \tag{6.2}$$

where α_t is the evolution step.

Due to the lack of the formal expression of the policy π, we can not compute the gradient $\frac{\partial V(\pi)}{\partial \theta^i}$ directly. Instead we use its estimates $\Delta V_{\theta^i}(\pi)$, which can be obtained from samplings on the policy distribution. To avoid large variance occurs in the learning in practical applications, we use the direction of the estimated gradient in the parameter update equation (6.2):

$$\theta_{t+1}^i = \theta_t^i + \alpha_t \frac{\Delta V_{\theta^i}(\pi)}{|\Delta V_{\theta^i}(\pi)|} = \theta_t^i + \alpha_t \eta_t^i \tag{6.3}$$

where $\eta_t^i = \frac{\Delta V_{\theta^i}(\pi)}{|\Delta V_{\theta^i}(\pi)|}$ is the direction of the estimated gradient.

Assume that at episode t the policy is π_t and the parameter vector of π_t is Θ_t. To update Θ_t by Equation (6.3), we introduce terms *direction payoffs* D_t^{+i}, D_t^{0i} and D_t^{-i} to indicate the accumulated payoff in the updating direction "positive", "none" and "negative". They are updated with a discount rate β as follows:

$$D_t^{pi} = \beta D_t^{pi} + (1-\beta) g_t^{pi}, p \in \{+,0,-\} \tag{6.4}$$

where g_t^{pi} is the virtual payoff in the updating direction p of parameter θ^i. It is obtained from sampling by the following process.

Starting from Θ_t, we randomly generate m parameter trials $\{\Theta_t^1, \ldots, \Theta_t^m\}$ around Θ_t by using the perturbation $\Theta_t^j = \Theta_t + \Delta\Theta_t^j$. $\Delta\Theta_t^j$ is defined as follows:

$$\begin{array}{c} \Delta\Theta_t^j = \{0^{j,1}, \ldots, 0^{j,n-1}, \Delta\theta_t^{j,n}, 0^{j,n+1}, \ldots, 0^{j,N}\} \\ (j = 1...m), n = random(1, N) \end{array} \tag{6.5}$$

where superscript j, i denotes the perturbation for ith parameter in jth trial. $random(1, N)$ generates a random integral number between 1 and N in the uniform distribution.

To eliminate the interaction between the perturbations of two parameters, only one parameter is chosen to have the perturbation in each trial. Now, there are m policies close to the initial policy $\pi_t = f(\Theta_t)$: $\{\pi_t^1, \ldots, \pi_t^m\}$. Note that $\Delta\theta_t^{j,n}$ is chosen randomly to be either $+\varepsilon\theta_t^n$ or $-\varepsilon\theta_t^i$. The perturbation step ε is currently fixed for all parameters. Each trial is repeated k times to get the average of payoffs as the expectation value, i.e. $E[r_t^j]$. $E[r_t^j]$ is accumulated together to get the payoff sum S_t^{+i} and S_t^{-i} according to n and $\Delta\theta_t^{j,n}$ as follows:

$$\begin{cases} S_t^{+i} = S_t^{+i} + E[r_t^j], \text{ if } n = i \text{ and } \Delta\theta_t^{j,n} = +\varepsilon\theta_t^n \\ S_t^{-i} = S_t^{-i} + E[r_t^j], \text{ if } n = i \text{ and } \Delta\theta_t^{j,n} = -\varepsilon\theta_t^n \end{cases} \tag{6.6}$$

Then we compute the average payoffs A_t^{+i} and A_t^{-i} for S_t^{+i} and S_t^{-i} respectively. Without any perturbation, we apply the policy π_t by k times and get $A_t^{0i} = E[r_t]$. Now, the g_t^{pi} is calculated by the following rule:

$$g_t^{pi} = \begin{cases} 1 & \text{if} A_t^{pi} = \max\{A_t^{+i}, A_t^{0i}, A_t^{-i}\} \\ 0 & otherwise \\ -1 & \text{if} A_t^{pi} = \min\{A_t^{+i}, A_t^{0i}, A_t^{-i}\} \end{cases}, p = \{+,0,-\} \tag{6.7}$$

And η_t^i is calculated by the following rule:

$$\eta_t^i = \begin{cases} 1 & \text{if} D_t^{+i} = \max\{D_t^{+i}, D_t^{0i}, D_t^{-i}\} \\ 0 & \text{if} D_t^{0i} = \max\{D_t^{+i}, D_t^{0i}, D_t^{-i}\} \\ -1 & \text{if} D_t^{-i} = \max\{D_t^{+i}, D_t^{0i}, D_t^{-i}\} \end{cases} \tag{6.8}$$

Once we get g_t^{pi}, D_t^{pi} is updated by Equation (6.4)

Finally, the parameters are updated by (6.3). The updated parameters construct an updated policy π_{t+1} which is the base point of the learning in the next episode $t+1$. The learning will be terminated when t is larger than episode limitation T_E or the termination condition of Equation (6.9) is satisfied in recent l steps ($l > 4$).

$$|E[r_t] - E[r_{t-1}]| < \tau \tag{6.9}$$

To speed up the learning process, an adaptive rate α_t is adopted here to replace the fixed α_t. It is adjusted according to the changing of $E[r_t]$. Suppose that the termination condition (6.9) is satisfied in the latest h episodes ($h < l$), α_t is adjusted as follows:

$$\alpha_t = \begin{cases} \lambda_1 \alpha_{t-1} \; if \, h = l-2 \\ \lambda_2 \alpha_{t-1} \; if \, h = l-3 \\ \lambda_3 \alpha_{t-1} \; if \, h = l-4 \end{cases} \tag{6.10}$$

where $0 < \lambda_1 < \lambda_2 < \lambda_3 < 1$. They are chosen arbitrarily as 0.7, 0.8, 0.9 representatively.

In this way, a larger learning rate can be used at the beginning and the oscillation at the later stage of the learning could be reduced. Algorithm 6.1 shows the policy gradient reinforcement learning algorithm that we have designed for the parameter updating of robotic fish swim patterns.

Algorithm 6.1 The policy gradient reinforcement learning algorithm

```
1.  Initialize: $\Theta = \Theta_0$, $D_t^{pi} = 0$
2.  while $t <= T_E$ do
3.    generate $m$ trial policies $\pi_t^j$ by perturbation (6.5) and reset $S_t^{+i}$, $S_t^{-i}$ to 0;
4.    repeat $\pi_t^j$ on Aifi for $k$ times, get $E[r_t^j]$;
5.    classify $E[r_t^j]$, get payoff sum $S_t^{+i}$, $S_t^{-i}$ and average payoff $A_t^{+j}$, $A_t^{-j}$ by (6.6);
6.    get $A_t^{0i}$ by make trial of $\pi_t$ without perturbation;
7.    calculate $g_t^{pi}$ and $\eta_t^i$ by (6.7) and (6.8);
8.    $\theta_{t+1}^i \leftarrow \theta_t^i + \alpha_t \eta_t^i$
9.    get $h$, where the condition (6.9) is satisfied in latest $h$ episodes;
10.    if $h < l$ then update $\alpha_t$ by (6.10)
11.    else terminate the learning process.
12.    endif
13.    $t = t + 1$;
14. endwhile (end of one episode)
```

6.4 State-based Reinforcement Learning in Cognitive Layer

In the cognitive layer, behaviours should be coordinated to achieve specific tasks, i.e. the fish should reason or plan its actions according to its current states. A typical RL based planner can be described by three parts: *Action Space*- a set of possible actions, *State Space*- the discrete possible situations of a robot on the way from its initial place to a goal, and a mapping from the state space to the action space. A *Markov Decision Process(MDP) Model* can be used to formally model such a planner. The RL can be used to learn the mapping function in this model.

6.4.1 Action Space and State Space

The cognitive layer aims at organizing activities to accomplish a task. The task could be turning on/off a software switch to the execution of a behaviour or just setting a configuration parameter of a behaviour. Actions in the cognitive layer are denoted as ca. To simplify the complexity, the action space is divided into two independent subspaces: level-plane actions and depth-control actions. The level-plane actions (la_i) is related with all the behaviours which affect the movement in all 2D planes parallelling water surface, for example *follow-wall* behaviour, while depth-control actions va_i can change the parameters of the behaviours which control the swimming depth, for example *keep-level* behaviour. Note that, *avoid-obstacle* behaviour is not listed as one of actions in the cognitive layer because it is a low-level behaviour in the behaviour layer.

The states in the cognitive layer are extracted from the sensor readings. They also include the information about which swim patterns is previously executed. The states come from the sensor readings but don't represent the quantity of individual sensors. They are the high-level condition or mode of Aifi. They reflect the significant physical events which are sensed or recognised by a temporal and spatial combination of several sensors. For example, if the down-facing infrared sensor outputs a higher value and the pressure sensor is larger than a threshold value, they indicate that the fish is near to the bottom of the tank. A set of these kinds of events constitutes the state space. In addition, an event can also be a previous swim pattern.

Formally, each event is denoted by a binary variant bv. Once an event occurs, the related bv is set to 1, otherwise it is clear to 0. Grouping n events $(bv_1, ..., bv_n)$ in an order generates a state cs for the cognitive layer, i.e. $cs \leftarrow bv_1, ..., bv_n$. The state space consists of a 2^n combination of n events. To decrease the size of the state space, these events are divided into two independent subspaces: level-states cs_l which only have relationship with the level actions, and vertical-states cs_v which are connected to the depth control actions.

6.4.2 Markov Decision Process Model

The RL based planner in the cognitive layer can be described by two finite MDPs: a level-MDP $\varGamma_l$ and a vertical-MDP $\varGamma_v$. The former is a model for level-states and level-actions while the latter is for vertical-states and depth-control actions. Assume that $\varGamma_l$ is defined by level-states cs_l, level-actions la and the one-step dynamics of the environment. The state transitions are described by transition probabilities:

$$P_l = Pr\{cs_{l(t+1)} | cs_{l(t)}, la_{(t)}\} \tag{6.11}$$

The expected value of the next reward given current state and action, $cs_{l(t)}$ and $la_{(t)}$, together with next state, $cs_{l(t+1)}$ is expressed as:

$$R_l = E\{r_{t+1}|cs_{l(t)}, la_{(t)}, cs_{l(t+1)}\} \tag{6.12}$$

A policy π is a mapping from states to actions. The optimal policy π^* maximizes the probability of reaching the goal. The *value* of a state s under a policy π, denoted $V^\pi(s)$, is the expected return when starting in s and following π thereafter. Function V^π is called *state-value function* for policy π. So, for Γ_l, we define $V_l^\pi(cs_l)$ as:

$$V_l^\pi(cs_l) = E_\pi\{R_t|cs_l\} = E_\pi\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \, |cs_l\} \tag{6.13}$$

where $E_\pi\{\}$ represents the expected value given that Aifi follows policy π. At the same time, the value of taking level action la in level-state cs_l is defined under policy π as $Q_l^\pi(cs_l, la)$:

$$Q_l^\pi(cs_l, la) = E_\pi\{R_t|cs_l, la\} = E_\pi\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \, |cs_l, la\} \tag{6.14}$$

For Γ_v, there are similar definitions of P_v, R_v, $V_v^\pi(cs_v)$ and $Q_v^\pi(cs_v, va)$. Given P_l and R_l for all level states cs_l and level actions la, a full description of Γ_l can be obtained. The optimal policy π^* can be found analytically by using *Dynamic Programming* which recursively calculates V_l^π and Q_l^π. If P_l is unknown, modelling techniques can be used to find it by the model-based RL. Alternatively, π^* can be found directly based on R_l through the model-free RL, such as *Monte Carlo method* or *Temporal-Difference Learning (TD Learning)*. In this chapter, *Q-learning* is designed to learn the mapping in the cognitive layer.

The *one-step Q-learning* updated function is as follows [9]:

$$Q(s, a) = Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right] \tag{6.15}$$

where Q represents Q_l^π, s denotes cs_l and a denotes la. α and γ are learning ratios. r is the observed reward when taking action a. s' is the succeeded state after taking action a. ϵ-greedy method is used to generate a from s using the policy derived from Q. A brief process is shown in Algorithm 6.2.

6.5 Experimental Results

6.5.1 Policy Gradient Learning for *Sharp-Turning* Swim Pattern

To prove the feasibility of the policy gradient learning algorithm, Aifi is used to learn the control parameters of the maximum turning angle of *sharp-turning*

Algorithm 6.2 The Q-learning algorithm

```
1. **Initialize:** $Q(s_0, a_0)$ arbitrarily;
2. **While** $t < T_E$ **do**:
3.   choose $a_t$ from $s_t$ using the policy derived from $Q$ via $\epsilon$-greedy;
4.   Take action $a_t$, observe new state $s_{t+1}$ and reward $r_{t+1}$;
5.   $Q(s_t, a_t) = Q(s_t, a_t) + \alpha \left[ r_t + 1 + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$;
6.   t=t+1;
7. **endwhile**
```

swim pattern. The *sharp-turning* swim pattern was designed in [7] where 8 key parameters $(\theta_1, \ldots, \theta_8)$ are extracted to mimic the sharp turning of real fish. Although this pattern has a kinematic function and a proximate dynamic function, it is quite difficult to obtain the analytical expression of the turning angle according to θ_i. So the relationship between the turning angle and θ_i is viewed as a blackbox which is described by a policy π with parameters θ_i. The objective of the policy gradient learning here is to find a local optimized policy π^* with which Aifi could be expected to have a largest turning angle by executing the *sharp-turning* swim pattern.

First, we adjusted θ_i manually by prior knowledge. The manual tuned θ_i are set as the initial value Θ_0 of the policy gradient learning algorithm, i.e. $\Theta_0 = \{\theta_1, \ldots, \theta_8\}$. Then the algorithm is started from Θ_0 and follows the step listed in Algorithm 6.1. The turning angle during sharp turning is measured by a compass sensor in the fish head. The instant reward is set to equal the final turning angle once *sharp-turning* finishes. In each episode, $m = 30$ trials are tested and each trial is repeated $k = 3$ times. Figure 6.3 shows the learning result for each episode. Initially the average turning angle is about 50 degrees. After about 90 hours learning, Aifi tried 4500 hard turnings, i.e. at the end of the 50th episode, the turning angle increases to 110 degree. Figure 6.4 shows a video sequence of *sharp-turning* after learning.

6.5.2 Q-learning for *Tank Border Exploration* Task

To test the feasibility of the proposed state based learning in the cognitive layer, a tank border exploration task is implemented by Aifi. The objective is to make Aifi to autonomously explore the tank border. It should follow tank walls in an appointed distance, be able to avoid the corner and other fishes, and keep itself in a desired depth level. Additionally it must keep the wall at its right side. Aifi is supposed to know nothing about its environment. It will learn to explore the tank border by the state based learning from scratch. The action space is structured according to the objective of the task. Four behaviours are chosen and customized from the generic behaviours layer for the level plane action subspace.

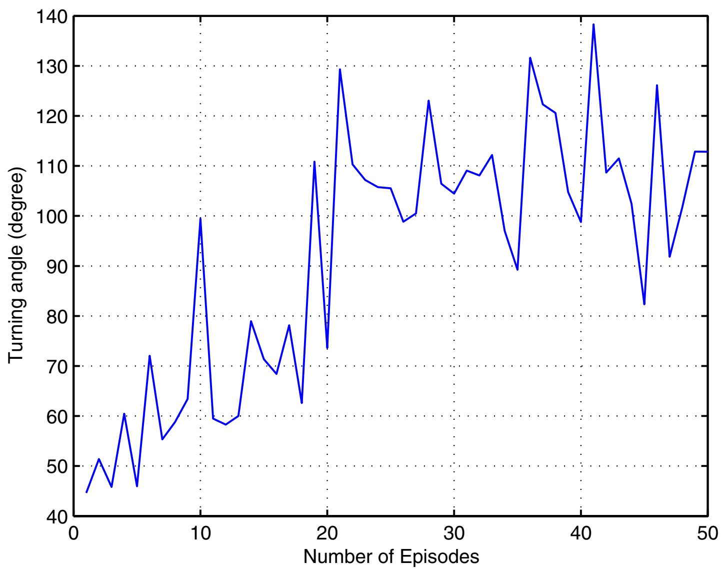

Fig. 6.3. The turning angle of *sharp-turning* swim pattern during learning.

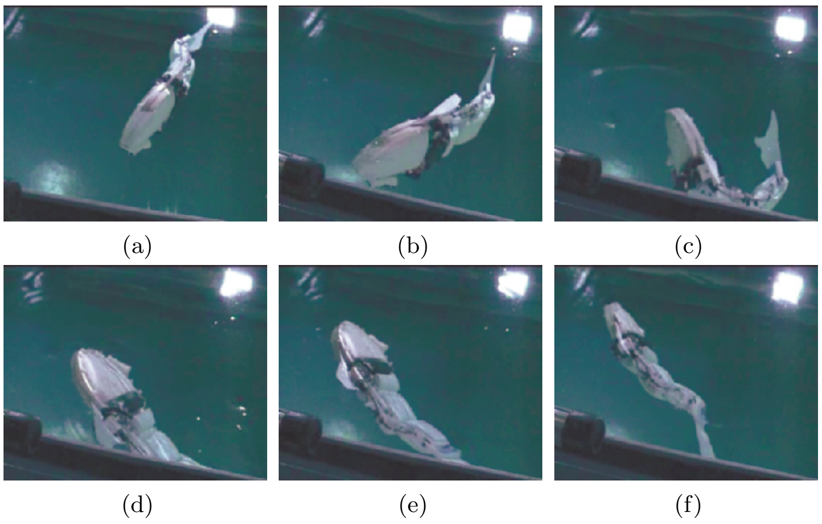

Fig. 6.4. A sharp-turning sequence of Aifi after learning

- *Wander* ($la_1 = WD$): This behaviour is limited on a 2D plane. Aifi randomly selects one of swim patterns from *cruise-straight* and *cruise-in-left/right-turning* to execute.

- *Follow-wall* ($la_2 = FLW$): This behaviour inherits from the *follow-wall* behaviour in the generic behaviour layer. The wall is appointed on the right of Aifi
- *Avoid-obstacle* ($la_3 = AO$): It is the same as the definition of *avoid-obstacle* behaviour in the generic behaviour layer except that it is limited in a 2D plane for the task.

Three events are defined to create the level states(cs_l):

- Is the wall on the right side of Aifi? (bv_1): It is decided by the right, front and left infrared sensors. $bv_1 = 1$ if recent history of these sensors satisfied some conditions.
- Is Aifi in a reasonable range from wall? (bv_2): This event is similar to bv_1 but it has more strict conditions.
- Is the wall on the left side of Aifi (bv_3): Like bv_1, it recognises the situation that the nearest wall is on the left side of fish. In other words, the swimming direction is reversed to the desired direction.

Because $bv_1 = 1$ and $bv_3 = 1$ are mutually exclusive, and so are $bv_1 = 0$ and $bv_2 = 1$, the total number of states is decreased from $8(2^3)$ to 4 as shown in Table 6.1

state cs_l	bv_1	bv_2	bv_3
0	0	N/A	0
1	0	N/A	1
2	1	0	N/A
3	1	1	N/A

Table 6.1. The states generated by events

There is no vertical action subspace in the cognitive layer for this task because the *keep-level* behaviour is capable enough for the task.

In this task, $r = 1$ when the fish in the *follow-wall* behaviour keeps following the wall, $r = -1$ when the fish in the *follow-wall* behaviour loses the wall to follow, $r = -3$ when there is a bumping between the wall and Aifi and $r = 0$ for other situations. According to Algorithm 6.2, policy π^* of the task is learned in a 3D robotic fish simulator [6] and then applied to Aifi. Figure 6.5 presents fish swimming trajectories during learning. It is clearly shown that Aifi is able to keep itself in a proper distance away from the wall after learning. Figure 6.6 gives the history of root mean square (RMS) errors between the learning trajectory and the desired path over 300 trips. The RMS error decreases and converges to a low value as the number of learning trips increases.

Figure 6.7 shows the trajectory of Aifi operated in the real tank, which is recorded from an overhead camera. The arrows in the figure show the heading directions and the circle points indicate the position of Aifi. The time step between two record points is 3 seconds. After Aifi is put into water from the point S with heading direction B_0, it selects the *find-wall* behaviour and executes it until reaching the left wall. Then the *follow-wall* behaviour is triggered. During its swimming, Aifi encounters an obstacle around point B_1. It implements a *sharp-left-turning* swim pattern to avoid it. Then it finds the wall again by the *cruise-in-turning-right* swim pattern. Finally, Aifi spent 105 seconds to swim around the tank one circle and finally reached the point E.

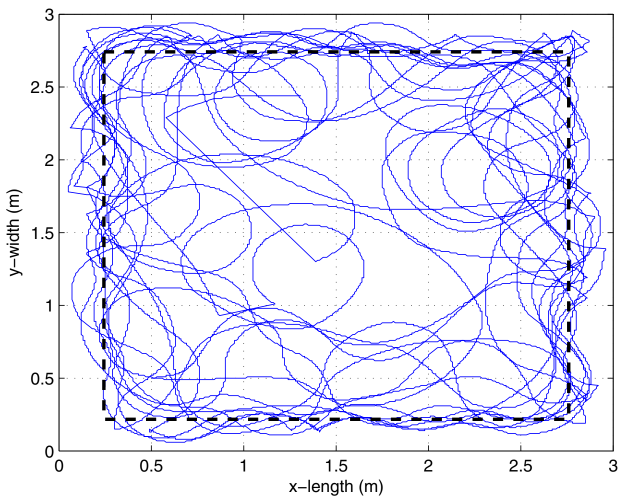

Fig. 6.5. The fish trajectory during learning. Note that the dashed line is the desired path, $\alpha = 0.5$ $\gamma = 0.3$, $\epsilon = 0.01$

6.6 Summary

In this chapter, reinforcement learning is used as learning methods in a layered control architecture of our robotic fish, Aifi. The swim pattern is learned by a sample-based policy gradient learning algorithm in the *swim pattern* layer. The task planning is learned by a state-based RL learning algorithm in the *cognitive* layer. The experimental tests show good performance of both algorithms. In the next step, we will apply reinforcement learning to learning of behaviours in the *behaviour* layer.

References

1. http://www.gumstix.org.

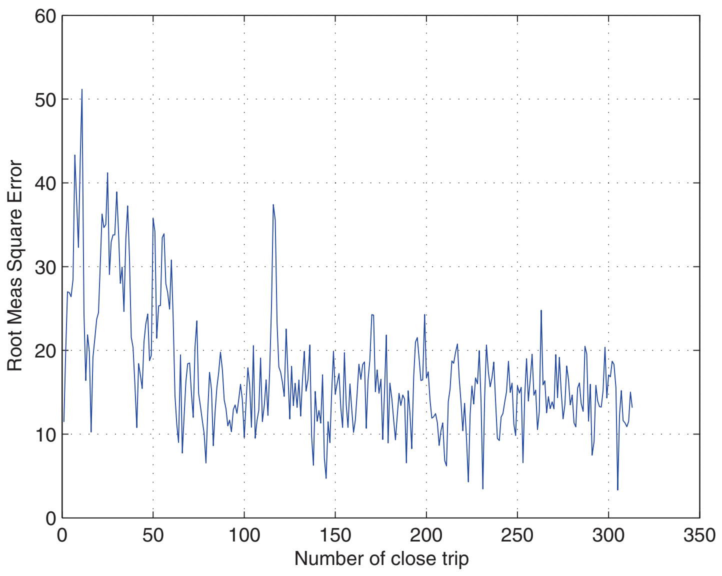

Fig. 6.6. The root mean square errors between the learning trajectory and the desired path against the number of learning trips. The errors are the sum of two parts: position error and heading error. Each trip has 50 steps.

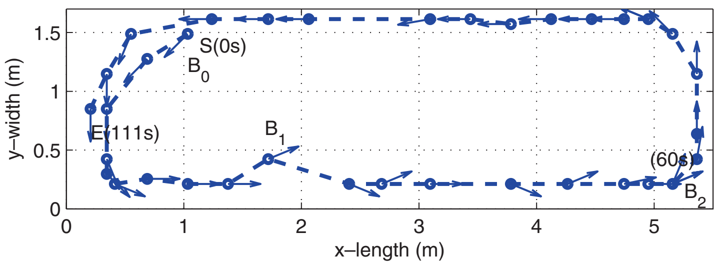

Fig. 6.7. A circular trajectory in the level plane

2. L. C. Barid and A. W. Moore. Gradient descent for general reinforcement learning. In *Proceedings of the International Conference on Advances in neural information processing systems II*, pages 968–974. MIT Press, 1999.
3. G. Z. Grudic, V. Kumar, and L. Ungar. Using policy gradient reinfrocement learning on autonmous robot controllers. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 406–411, Las Vagas, Navada, USA, Oct 2003.
4. G. Hornby, S. Takamura, J. Yokono, O. Hanagata, T. Yamamoto, and M. Fujita. Evolving robust gaits with AIBO. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 3040–3045, 2000.
5. N. Kohl and P. Stone. Policy gradient reinforcement learning for fast quadrupedal locomotion. In *Proceedings of IEEE International Conference on Robotics and Automation*, volume 3, pages 2619–2624, May 2004.

6. J. Liu and H. Hu. Building a 3d simulator for autonomous navigation of robotic fishes. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 613–618, Sendai, Japan, Oct 2004.
7. J. Liu and H. Hu. Mimicry of sharp turning behaviours in a robotic fish. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 3329–3334, Barcelona, Spain, April 2005.
8. L. Peshkin, K. Kim, N. Meuleau, and L. Kaelbling. Learning to cooperate via policy search. In *Proceedings of the 6th International Conference on Uncertainty in Artificial Intelligence*, pages 307–314, 2000.
9. C. J. C. H. Watkins. *Learning from Delayed Rewards.* PhD thesis, Cambridge University, 1989.
10. R. J. William. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.
11. R. Zhang and P. Vadakkepat. An evolutionary algorithm for trajectory based gait generation of biped robot. In *Proceedings of the International Conference on Computational Intelligence, Robotics and Autonomous Systems*, Singapore, 2003.