# Autonomous Learning of Ball Trapping in the Four-legged Robot League

Hayato Kobayashi[1], Tsugutoyo Osaki[2], Eric Williams[2], Akira Ishino[3], and Ayumi Shinohara[2]

[1] Department of Informatics, Kyushu University, Japan
[2] Graduate School of Information Science, Tohoku University, Japan
[3] Office for Information of University Evaluation, Kyushu University, Japan
{h-koba@i,ishino.uoc@mbox.nc}.kyushu-u.ac.jp,
{osaki,ayumi}@shino.ecei.tohoku.ac.jp, eaw@ucla.edu

**Abstract.** This paper describes an autonomous learning method used with real robots in order to acquire ball trapping skills in the four-legged robot league. These skills involve stopping and controlling an oncoming ball and are essential to passing a ball to each other. We first prepare some training equipment and then experiment with only one robot. The robot can use our method to acquire these necessary skills on its own, much in the same way that a human practicing against a wall can learn the proper movements and actions of soccer on his/her own. We also experiment with two robots, and our findings suggest that robots communicating between each other can learn more rapidly than those without any communication.

## 1 Introduction

For robots to function in the real world, they need the ability to adapt to unknown environments. These are known as *learning* abilities, and they are essential in taking the next step in RoboCup. As it stands now, it is humans, not the robots themselves, that hectically attempt to adjust programs at the competition site, especially in the real robot leagues. But what if we look at RoboCup in a light similar to that of the World Cup? In the World Cup, soccer players can practice and confirm certain conditions on the field before each game. In making this comparison, should robots also be able to adjust to new competition and environments on their own? This ability for something to learn on its own is known as *autonomous learning* and is regarded as important.

In this paper, we force robots to autonomously learn the basic skills needed for passing to each other in the four-legged robot league. Passing (including receiving a passed ball) is one of the most important skills in soccer and is actively studied in the simulation league. For several years, many studies [1, 2] have used the benchmark of good passing abilities, known as "keepaway soccer", in order to learn how a robot can best learn passing. However, it is difficult for robots to even control the ball in the real robot leagues. In addition, robots in the four-legged robot league have neither a wide view, high-performance camera, nor laser range finders. As is well known, they are not made for playing soccer. Quadrupedal locomotion alone can be a difficult enough challenge. Therefore, they must improve upon basic skills in order to solve these difficulties, all before pass-work learning can begin. We believe that basic skills should be learned by a real robot, because of the necessity of interaction with a real environment. Also, basic skills should be autonomously learned because changes to an environment will always consume much of people's time and energy if the robot cannot adjust on its own.

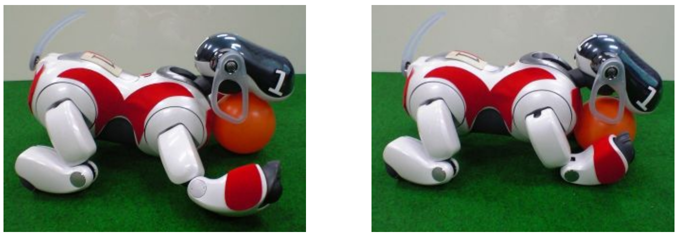
There have been many studies conducted on the autonomous learning of quadrupedal locomotion, which is the most basic skill for every movement. These studies began as far back as the beginning of this research field and continue still today [3–6]. However, the skills used to control the ball are often coded by hand and have not been studied as much as gait learning. There also have been several similar works related to how robots can learn the skills needed to control the ball. Chernova and Veloso [7] studied the learning of ball kicking skills, which is an important skill directly related to scoring points. Zagal and Solar [8] studied the learning of kicking skills as well, but in a simulated environment. Although it was very interesting in the sense that robots could not have been damaged, the simulator probably could not produce complete, real environments. Fidelman and Stone [9] studied the learning of ball acquisition skills, which are unique to the four-legged robot league. They presented an elegant method for autonomously learning these unique, advanced skills. However, there has thus far been no study that has tried to autonomously learn the stopping and controlling of an oncoming ball, i.e. *trapping* the ball. In this paper, we present an autonomous learning method for ball trapping skills. Our method will enhance the game by way of learned pass-work in the four-legged robot league.

The remainder of this paper is organized as follows. In Section 2, we begin by specifying the actual, physical actions used in trapping the ball. Then we simplify the learning process for ball trapping down to a one-dimensional model, and finally, we illustrate and describe our training equipment used by the robots while training in solitude. In Section 3, we formalize a learning problem and show our autonomous learning algorithm for it. In Section 4, we experiment using one robot, two robots, and two robots with communication. Finally, Section 5 presents our conclusions.

## 2 Preliminary

### 2.1 Ball Trapping

Before any learning can begin, we first have to accurately create the appropriate physical motions to be used in trapping a ball accurately before the learning process. The picture in Fig. 1 (a) shows the robot's pose at the end of the motion. The robot begins by spreading out its front legs to form a wide area with which to receive the ball. Then, the robot moves its body back a bit in order to absorb the impact caused by the collision of the body with the ball and to reduce the rebound speed. Finally, the robot lowers its head and neck, assuming that the ball has passed below the chin, in order to keep the ball from bouncing off of its chest and away from its control. Since the camera of the robot is equipped on the tip of the nose, it actually cannot watch the ball below the chin. This series of motions is treated as single motion, so we can neither change the speed of the motion, nor interrupt it, once it starts. It takes 300 ms (= 60 steps $\times$ 5 ms) to perform. As opposed to grabbing or grasping the ball, this trapping motion is instead



(a) trapping motion (b) pre-judgment motion

**Fig. 1.** The motion to actually trap the ball (a), and the motion to judge if it succeeded in trapping the ball (b).

thought of as keeping the ball, similar to how a human player would keep control of the ball under his/her foot.

The judgment of whether the trap succeeded or failed is critical for autonomous learning. Since the ball is invisible to the robot's camera when it's close to the robot's body, we utilized the chest PSD sensor. However, the robot cannot make an accurate judgment when the ball is not directly in front of their chest or after it takes a droopy posture. Therefore, we utilized a "pre-judgment motion", which takes 50 ms (= 10 steps $\times$ 5 ms), immediately after the trapping motion is completed, as shown in Fig. 1 (b). In this motion, the robot fixes the ball between its chin and chest and then lifts its body up slightly so that the ball will be located immediately in front of the chest PSD sensor, assuming the ball was correctly trapped to begin with.

### 2.2 One-dimensional Model of Ball Trapping

Acquiring ball trapping skills in solitude is usually difficult, because robots must be able to search for a ball that has bounced off of them and away, then move the ball to an initial position, and finally kick the ball again. This requires sophisticated, low-level programs, such as an accurate, self-localization system; a strong shot that is as straight as possible; and a locomotion which utilizes the odometer correctly. In order to avoid additional complications, we simplify the learning process a bit more.

First, we assume that the passer and the receiver face each other when the passer passes the ball to the receiver, as shown Fig. 2. The receiver tries to face the passer while watching the ball that the passer is holding. At the same time, the passer tries to face the receiver while looking at the red or blue chest uniform of the receiver. This is not particularly hard to do, and any team should be able to accomplish it. As a result, the robots will face each other in a nearly straight line. The passer need only shoot the ball forward so that the ball can go to the receiver's chest. The receiver, in turn, has only to learn a technique for trapping the oncoming ball without it bouncing away from its body.

Ideally, we would like to treat our problem, which is to learn ball trapping skills, one-dimensionally. In actuality though, the problem cannot be fully viewed in one-dimension, because either the robots might not precisely face each other in a straight line, or because the ball might curve a little due to the grain of the grass. We will discuss this problem in Section 5.

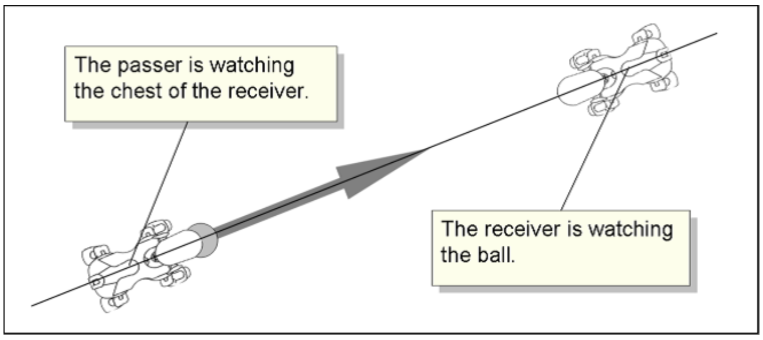

The passer is watching the chest of the receiver.
The receiver is watching the ball.

**Fig. 2.** One-dimensional model of ball trapping problem.

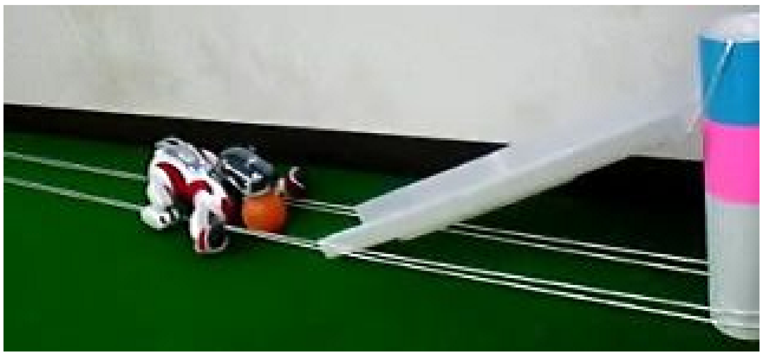


**Fig. 3.** Training equipment for learning ball trapping skills.

### 2.3 Training Equipment

The equipment we prepared for learning ball trapping skills in one-dimensional is fairly simple. As shown in Fig. 3, the equipment has rails of width nearly equal to an AIBO's shoulder-width. These rails are made of thin rope or string, and their purpose is to restrict the movement of the ball, as well as the quadrupedal locomotion of the robot, to one-dimension. Aside from these rails, the robots use a slope placed at the edge of the rail when learning in solitude. They kick the ball toward the slope, and they can learn trapping skills by trying to trap the ball after it returns from having ascended the slope.

## 3 Learning Method

Fidelman and Stone [9] showed that the robot can learn to grasp a ball. They employed three algorithms: hill climbing, policy gradient, and amoeba. We cannot, however, directly apply these algorithms to our own problem because the ball is moving fast in our case. It may be necessary for us to set up an equation which incorporates the friction of the rolling ball and the time at which the trapping motion occurs if we want to view our problem in a manner similar to these parametric learning algorithms. In this paper, we apply reinforcement learning algorithms [10]. Since reinforcement learning requires no background knowledge, all we need to do is give the robots the appropriate reward for a successful trapping so that they can successfully learn these skills.

The reinforcement learning process is described as a sequence of states, actions, and rewards

$$s_0, a_0, r_1,\ s_1, a_1, r_2,\ \ldots, s_i, a_i, r_{i+1},\ s_{i+1}, a_{i+1}, r_{i+2},\ \ldots,$$

which is a reflection of the interaction between the learner and the environment. Here, $s_t \in S$ is a state given from the environment to the learner at time $t$ ($t \geq 0$), and $a_t \in \mathcal{A}(s_t)$ is an action taken by the learner for the state $s_t$, where $\mathcal{A}(s_t)$ is the set of actions available in state $s_t$. One time step later, the learner receives a numerical reward $r_{t+1} \in \mathcal{R}$, in part as a consequence of its action, and finds itself in a new state $s_{t+1}$.

Our interval for decision making is 40 ms and is in synchronization with the frame rate of the CCD-camera. In the sequence, we treat each 40 ms as a single time step, i.e. $t = 0, 1, 2, \cdots$ means 0 ms, 40 ms, 80 ms, $\cdots$, respectively. In our experiments, the states essentially consist of the information on the moving ball: relative position to the robot, moving direction, and the speed, which are estimated by our vision system. Since we have restricted the problem to one-dimensional movement in Section 2.2, the state can be represented by a pair of scalar variables $x$ and $dx$. The variable $x$ refers to the distance from the robot to the ball estimated by our vision system, and $dx$ simply refers to the difference between the current $x$ and the previous $x$ of one time step before. We limited the range of these state variables such that $x$ is in [ 0 mm, 2000 mm ], and $dx$ in [ $-200$ mm, 200 mm ]. This is because if a value of $x$ is greater than 2000, it will be unreliable, and if the absolute value of $dx$ is greater than 200, it must be invalid in games (e.g. $dx$ of 200 mm means 5000 mm/s).

Although the robots have to do a large variety of actions to perform fully-autonomous learning by nature, as far as our learning method is concerned, we can focus on the following two macro-actions. One is *trap*, which initiates the trapping motions described in Section 2.1. The robot's motion cannot be interrupted for 350 ms until the trapping motion finishes. The other is *ready*, which moves its head to watch the ball and preparing to *trap*. Each reward given to the robot is simply one of $\{+1, 0, -1\}$, depending on whether it successfully traps the ball or not. The robot can make a judgment of that success by itself using its chest PSD sensor. The reward is 1 if the *trap* action succeeded, meaning the ball was correctly captured between the chin and the chest after the *trap* action. A reward of $-1$ is given either if the *trap* action failed, or if the ball touches the PSD sensor before the *trap* action is performed. Otherwise, the reward is 0. We define the period from kicking the ball to receiving any reward other than 0 as one *episode*. For example, if the current episode ends and the robot moves to a random position with the ball, then the next episode begins when the robot kicks the ball forward.

In summary, the concrete objective for the learner is to acquire the correct timing for when to initiate the trapping motion depending on the speed of the ball by trial and error. Fig. 4 shows the autonomous learning algorithm used in our research. It is a combination of the episodic SMDP Sarsa($\lambda$) with the linear tile-coding function approximation (also known as CMAC). This is one of the most popular reinforcement learning algorithms, as seen by its use in the keepaway learner [1].

```
while still not acquiring trapping skills do
    go get the ball and move to a random position with the ball;
    kick the ball toward the slope;
    $s \leftarrow$ a state observed in the real environment;
    forall $a \in \mathcal{A}(s)$ do
        $F_a \leftarrow$ set of tiles for $a, s$;
        $Q_a \leftarrow \sum_{i \in F_a} \theta(i)$;
    end
    $lastAction \leftarrow$ an optimal action selected by $\epsilon$-greedy;
    $\vec{e} \leftarrow 0$;
    forall $i \in F_{lastAction}$ do $e(i) \leftarrow 1$;
    $reward \leftarrow 0$;
    while $reward = 0$ do
        do $lastAction$;
        if $lastAction = trap$ then
            if the ball is held then $reward \leftarrow 1$;
            else $reward \leftarrow -1$;
        else
            if collision occurs then $reward \leftarrow -1$;
            else $reward \leftarrow 0$;
        end
        $\delta \leftarrow reward - Q_{lastAction}$;
        $s \leftarrow$ a state observed in the real environment;
        forall $a \in \mathcal{A}(s)$ do
            $F_a \leftarrow$ set of tiles for $a, s$;
            $Q_a \leftarrow \sum_{i \in F_a} \theta(i)$;
        end
        $lastAction \leftarrow$ an optimal action selected by $\epsilon$-greedy;
        $\delta \leftarrow \delta + Q_{lastAction}$;
        $\vec{\theta} \leftarrow \vec{\theta} + \alpha\delta\vec{e}$;
        $Q_{lastAction} \leftarrow \sum_{i \in F_{lastAction}} \theta(i)$;
        $\vec{e} \leftarrow \lambda\vec{e}$;
        if player acting in state $s$ then
            forall $a \in \mathcal{A}(s)$ s.t. $a \neq lastAction$ do
                forall $i \in F_a$ do $e(i) \leftarrow 0$;
            end
            forall $i \in F_{lastAction}$ do $e(i) \leftarrow 1$;
        end
    end
    $\delta \leftarrow reward - Q_{lastAction}$;
    $\vec{\theta} \leftarrow \vec{\theta} + \alpha\delta\vec{e}$;
end
```

**Fig. 4.** Algorithm of our autonomous learning (based on keepaway learner [1]).

Here, $F_a$ is a *feature set* specified by tile coding with each action $a$. In this paper, we use two-dimensional tiling and set the number of tilings to 32 and the number of tiles to about 5000. We also set the tile width of $x$ to 20 and the tile width of $dx$ to 50. The vector $\vec{\theta}$ is a *primary memory vector*, also known as a *learning weight vector*, and $Q_a$ is a *Q-value*, which is represented by the sum of $\vec{\theta}$ for each value of $F_a$. The policy *$\epsilon$-greedy* selects a random action with probability $\epsilon$, and otherwise, it selects the action with the maximum $Q$-value. We set $\epsilon = 0.01$. Moreover, $\vec{e}$ is an *eligibility trace*, which stores the credit that past action choices should receive for current rewards. $\lambda$ is a *trace-decay parameter* for the eligibility trace, and we simply set $\lambda = 0.0$. We set the *learning rate parameter* $\alpha = 0.5$ and the *discount rate parameter* $\gamma = 1.0$.

## 4 Experiments

### 4.1 Training Using One Robot

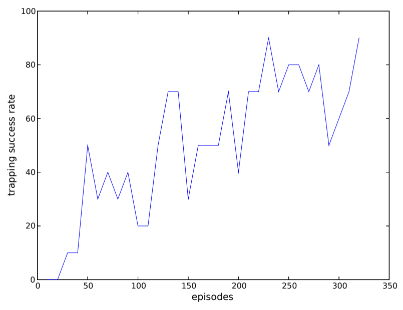
We first experimented by using one robot along with the training equipment that was illustrated in Section 2.3. The robot could train in solitude and learn ball trapping skills on its own.

Fig. 5(a) shows the trapping success rate, which is how many times the robot successfully trapped the ball in 10 episodes. It reached about 80% or more after 250 episodes, which took about 60 minutes using 2 batteries. Even if robots continue to learn, the success rate is unlikely to ever reach 100%. This is because the trapping motions, which force the robot to move slightly backwards in order to try and reduce the bounce effect, can hardly be expected to capture a slow, oncoming ball that stops just in front of it.

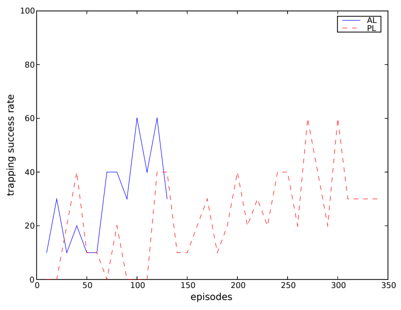
Fig. 6 shows the result of each episode by plotting a circle if it was successful, a cross if it failed in spite of trying to trap, and a triangle if it failed because of doing nothing. From the 1st episode to the 50th episode, the robots simply tried to trap the ball while it was moving with various velocities and at various distances. They made the mistake of trying to trap the ball even when it was moving away ($dx > 0$), because we did not give them any background knowledge, and we only gave them two variables: $x$ and $dx$. From the 51st episode to the 100th episode, they learned that they could not trap the ball when it was far away ($x > 450$) or when it was moving away ($dx > 0$). From the 101st episode to 150th episode, they began to learn the correct timing for a successful trapping, and from the 151st episode to 200th episode, they almost completely learned the correct timing.

### 4.2 Training Using Two Robots

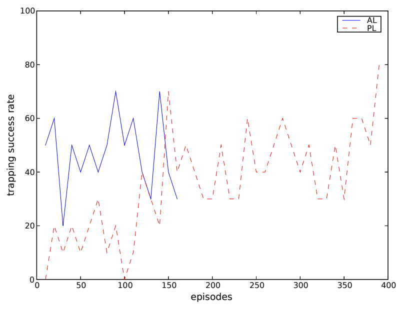
In the case of training using two robots, we simply replace the slope in the training equipment with another robot. We call the original robot the *Active Learner* (AL) and the one which replaced with slope the *Passive Learner* (PL). AL is the same as in case of training using one robot. On the other hand, PL differs from AL in that PL does not



(a) one robot



(b) two robots



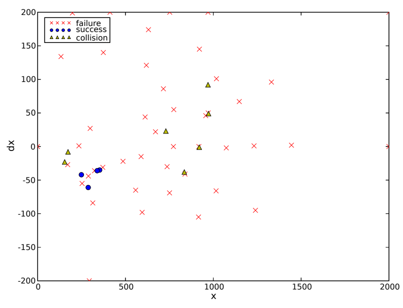
(c) two robots with communication

**Fig. 5.** Results of three experiments.

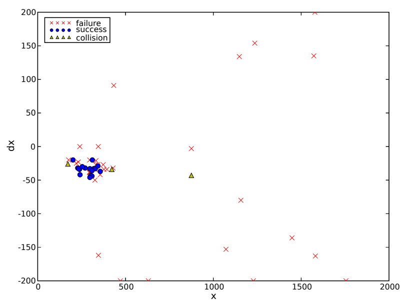
search out nor approach the ball if the trapping failed. Only AL does so. Other than this difference, PL and AL are basically the same.

We experimented for 60 minutes by using both AL and PL that had learned in solitude for 60 minutes using the training equipment. Theoretically, we would expect them to succeed in trapping the ball after only a short time. However, by trying to trap the ball while in obviously incorrect states, they actually failed repeatedly. The reason for this was because the estimation of the ball's distance to the robot-in-waiting became unreliable, as shown in Fig. 7. This, in turn, was due to the other robot holding the ball below its head before kicking it forward to its partner. Such problems can occur during the actual games, especially in poor lighting conditions, when teammates and adversaries are holding the ball.

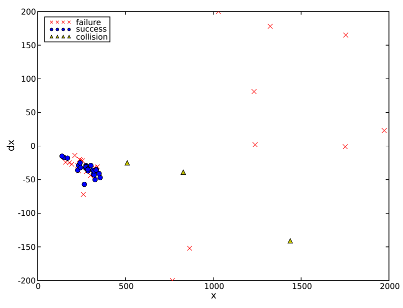
Although we are of course eager to overcome this problem, we should not force a solution that discourages the robots from holding the ball first, because ball holding skills help them to properly judge whether or not they can successfully trap the ball. It also serves another purpose, which is to give the robots a nicer, straighter kick. Moreover, there is no way we can absolutely keep the adversary robots from holding the ball. Although there are several solutions (e.g. measuring the distance to the ball by using green pixels or sending the training partner to get the ball), we simply continued to make the robots learn without having made any changes. This was done in an attempt



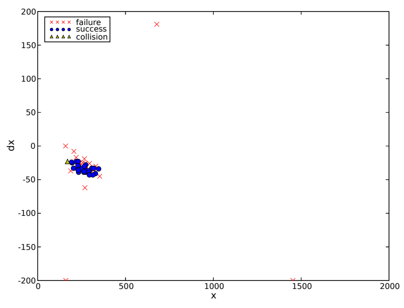
(a) Episodes 1–50



(b) Episodes 51–100

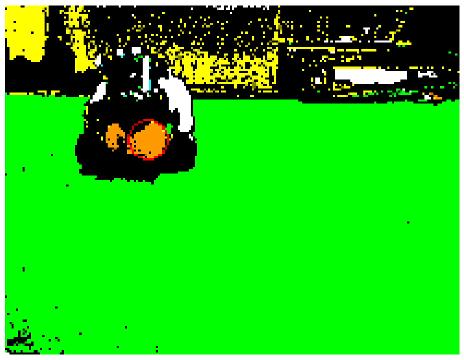


(c) Episodes 101–150



(d) Episodes 151–200

**Fig. 6.** Learning process from 1st episode to 200th episode. A circle indicates successful trapping, a cross indicates failed trapping, and a triangle indicates collision with the ball.



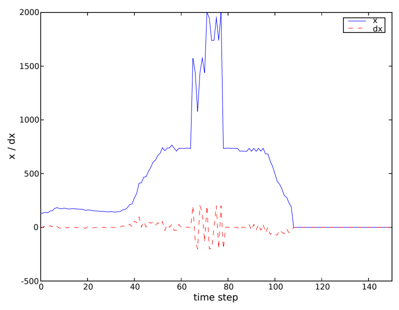


**Fig. 7.** The left figure shows how our vision system recognizes a ball when the other robot holds it. The ball looks to be smaller than it is, because a part of it is hidden by the partner and its shadow, resulting in an estimated distance to the ball that is further away than it really is. The right figure plots the estimated values of the both the distance $x$ and the velocity $dx$, when the robot kicked the ball to its partner, the partner trapped it, and then the partner kicked it back. When the training partner was holding the ball under its head though (the center of the graph), we can see the robot obviously miscalculated ball’s true distance.

to allow the robots to gain experience related to irrelevant states. In fact, it turns out they should never try to trap the ball when $x \geq 1000$ and $dx \geq 200$. Moreover, they should probably not try to trap the ball when $x \geq 1000$ and $dx \leq -200$.

Fig. 5(b) shows the results of training using two robots. They began to learn that they should probably not try to trap the ball while in irrelevant states, as this was a likely indicator that the training partner was in possession of the ball. This was learned quite slowly though, because the AL can only learn successful trapping skills when PL itself succeeds. If PL fails, AL's episode is not incremented. Even if the player nearest the ball can go get it, the problem is not resolved because then they just learn slowly in the end, though simultaneously.

### 4.3 Training Using Two Robots with Communication

Training using two robots, like in the previous section, unfortunately takes a long time to complete. In this section, we will look at accelerating their learning by allowing them to communicate with each other.

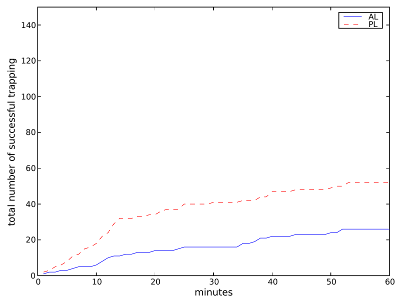
First, we made the robots share their experiences with each other, as in [11]. However, if they continuously communicated with each other, they could not do anything else, because the excessive processing would interrupt the input of proper states from the real-time environment. Therefore, we made the robots exchange their experiences, which included what action $a_t$ they performed, the values of the state variables $x_t$ and $dx_t$, and the reward $r_{t+1}$ at time $t$, but this was done only when they received a reward other than 0, i.e. the end of each episode. They then updated their $\overrightarrow{\theta}$ values using the experiences they received from their partner. As far as the learning achievements for our research is concerned, they can successfully learn enough using this method.

We also experimented in the same manner as Section 4.2 using two robots which can communicate with each other. Fig. 5(c) shows the results of this experiment. They could rapidly adapt to unforeseen problems and acquire practical trapping skills. Since PL learned its skills before AL learned, it could relay to AL the helpful experience, effectively giving AL about a 50% learned status from the beginning. These results indicate that the robots with communication learned more quickly than the robots without communication.

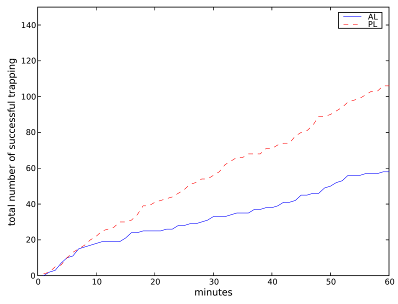
### 4.4 Discussion

The three experiments above showed that robots could efficiently learn ball trapping skills and that the goal of pass-work by robots can be achieved in one-dimension. In order to briefly compare those experiments, Fig. 8 presents a few graphs, where the $x$-axis is the elapsed time and the $y$-axis is the total number of successes so far. Fig. 8(a) and Fig. 8(b) shows the learning process with and without communication, respectively, for 60 minutes after pre-learning for 60 minutes by using two robots from the beginning. Fig. 8(c) and Fig. 8(d) shows the learning process with and without communication, respectively, after pre-learning for 60 minutes in solitude.

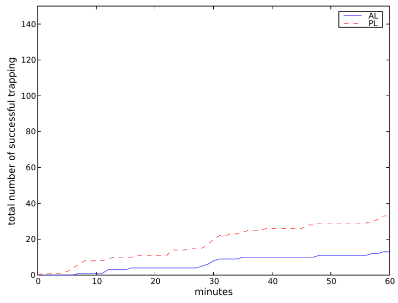
Comparing (a) and (c) with (b) and (d) has us conclude that allowing AL and PL to communicate with each other will lead to more rapid learning compared to no communication. Comparing (a) and (b) with (c) and (d), the result is different from our



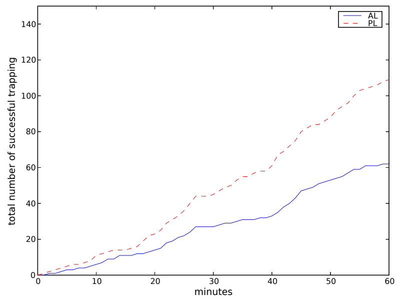
(a) without communication after pre-learning by using two robots



(b) with communication after pre-learning by using two robots



(c) without communication after pre-learning in solitude



(d) with communication after pre-learning in solitude

**Fig. 8.** Total numbers of successful trappings with respect to the elapsed time.

expectation. Actually, the untrained robots learned as much as or better than trained robots for 60 minutes. The trained robots seems to be over-fitted for slow-moving balls, because the ball was slower in the case of one robot learning than in the case of two due to friction on the slope. However, it is still good strategy to train robots in solitude at the beginning, because experiments that solely use two robots can make things more complicated. In addition robots should also learn the skills for a relatively slow-moving ball anyway.

## 5 Conclusions and Future Work

In this paper, we presented an autonomous learning method for use in acquiring ball trapping skills in the four-legged robot league. Robots could learn and acquire the skills without human intervention, except for replacing discharged batteries. They also successfully passed and trapped a ball with another robot and learn more quickly when exchanging experiences with each other. All movies of the earlier and later phases of our experiments are available on-line (http://www.jollypochie.org/papers/).

We also tried finding out whether or not robots can trap the ball without the use of the training equipment (rails for ball guidance). We rolled the ball to the robot by hand,

and the robot could successfully trap it, even if the ball moved a few centimeters away from the center of its chest. At the same time though, the ball would often bounce off of it, or the robot did nothing if the ball happened to veer significantly away from the center point. In the future, we plan to extend trapping skills into two-dimensions using layered learning [12], e.g. we will try to introduce three actions of staying, moving to the left, and moving to the right into higher-level layers. Since two-dimensions are essentially the same as one-dimension in this case, it may be possible to simply use a wide slope. Good two-dimensional trapping skills can directly make keepers or goalies stronger. In order to overcome the new problems associated with a better goalie on the opposing team, robots may have to rely on learning better passing skills, as well as learning even better ball trapping skills. A quick ball is likely to move straightforward with stability, but robots as they are now can hardly trap a quick ball. Therefore, robots must learn skills in shooting as well as how to move the ball with proper velocity. It would be most effective if they learn these skills alongside trapping skills. This is a path that can lead to achieving successful keepaway soccer [1] techniques for use in the four-legged robot league.

## References

1. Peter Stone, Richard S. Sutton, and Gregory Kuhlmann. Reinforcement learning for robocup soccer keepaway. *Adaptive Behavior*, 13(3):165–188, 2005.
2. William H. Hsu, Scott J. Harmon, Edwin Rodriguez, and Christopher Zhong. Empirical comparison of incremental reuse strtegies in genetic programming for keep-away soccer. In *Late Breaking Papers at the 2004 Genetic and Evolutionary Computation Conference*, 2004.
3. Gregory S. Hornby, Seichi Takamura, Takashi Yamamoto, and Masahiro Fujita. Autonomous evolution of dynamic gaits with two quadruped robots. *IEEE Transactions on Robotics*, 21(3):402–410, 2005.
4. Min Sub Kim and William Uther. Automatic gait optimisation for quadruped robots. In *Proceedings of 2003 Australasian Conference on Robotics and Automation*, pages 1–9, 2003.
5. Nate Kohl and Peter Stone. Machine learning for fast quadrupedal locomotion. In *The Nineteenth National Conference on Artificial Intelligence*, pages 611–616, 2004.
6. Joel D. Weingarten, Gabriel A. D. Lopes, Martin Buehler, Richard E. Groff, and Daniel E. Koditschek. Automated gait adaptation for legged robots. In *IEEE International Conference on Robotics and Automation*, 2004.
7. Sonia Chernova and Manuela Veloso. Learning and using models of kicking motions for legged robots. In *Proceedings of International Conference on Robotics and Automation*, 2004.
8. Juan Cristóbal Zagal and Javier Ruiz del Solar. Learning to kick the ball using back to reality. In *RoboCup 2004: Robot Soccer World Cup VIII*, volume 3276 of *LNAI*, pages 335–347. Springer-Verlag, 2005.
9. Peggy Fidelman and Peter Stone. Learning ball acquisition on a physical robot. In *2004 International Symposium on Robotics and Automation (ISRA)*, 2004.
10. Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.
11. R. Matthew Kretchmar. Parallel reinforcement learning. In *The 6th World Conference on Systemics, Cybernetics, and Informatics.*, 2002.
12. Peter Stone and Manuela M. Veloso. Layered learning. In *Proceedings of 11th European Conference on Machine Learning*, volume 1810, pages 369–381. Springer, Berlin, 2000.