Jolly Pochie
Technical Report in RoboCup 2006

Hayato Kobayashi [1] Jun Inoue [1] Tsugutoyo Osaki [2]
Tetsuro Okuyama [2] Shuhei Yanagimachi [2] Takahito Sasaki [2]
Keisuke Oi [2] Seiji Hirama [2] Akira Ishino [2]
Ayumi Shinohara [2] Akihiro Kamiya [2] Satoshi Abe [2]
Wataru Matsubara [2] Tomoyuki Nakamura [2]

[1] Department of Informatics, Kyushu University
[2] Department of System Information Sciences, GSIS, Tohoku University

January 15, 2007

Kyushu University and Tohoku University

Contents

Chapter 1

Introduction

The team "Jolly Pochie [dʐóli·pótʃi:]" has participated in the RoboCup Four-legged League since 2003. Jolly Pochie consists of the faculty staff and graduate/undergraduate students of Department of Informatics, Kyushu University and Department of System Information Sciences, GSIS, Tohoku University [12].

Faculty members
Ayumi Shinohara and Akira Ishino

Graduate Students
Hayato Kobayashi, Satoshi Abe, Akihiro Kamiya, Tsugutoyo Osaki and Tetsuro Okuyama

Undergraduate Students
Shuhei Yanagimachi, Keisuke Oi, Takahito Sasaki, Tomoyuki Nakamura, Seiji Hirama, Wataru Matsubara and Eric Williams

Our research interests mainly include machine learning, machine discovery, data mining, image processing, string processing, software architecture, visualization, and so on. RoboCup is a suitable benchmark problem for these domains.

Our main improvements of this year are broken into three parts. The first is more accurate object recognition of our vision system by improving our object recognition algorithms and our learning tool generating color tables. Our framework, our modules and our bots is the same as last year. They are explained in technical report of last year [16].

The second is more robust estimation of ball's location by utilizing a new localization technique. The third is work saving for developing successful ball trapping skills by utilizing an autonomous learning technique for the skills.

The rest of this report is organized as follows. Chapter 2, we give outline of our image processing system. Chapter 3 shows the ball localization techniques. Chapter 4 shows how to learn ball trapping skills. Chapter 5 describes our new behavior system, and Chapter 6 illustrates soccer strategies used in the system. Chapter 7 describes our new shot motions. Chapter 8 describes the results of the

technical challenges in RoboCup 2006. Finally, Chapter 9 presents the conclusion of this report.

Chapter 2

Vision

We use a color table for color segmentation of images. The color table consists of the 3D-Matrix in the YUV space, which size is $64 \times 64 \times 64$. The color table is made manually using a nearest neighbor learning algorithm. The details are written in our technical report of last year [16].

This year, we improved our learning tool showed in Fig. 2.1. Our learning process is as follows. First we select one of the symbolic colors (white, green, dark blue, light blue, orange, yellow, red, pink or black). Next we click a point in the raw image on the tool for labeling it as the selected symbolic color. In last year, it took much time for making color tables, because we were not able to select more than one point in the image. Therefore, we extended the function of the tool so that we can select multiple points at one time. In the new tool, we can label the points in the dragged range as the selected symbolic color with dragging a part of the image. The process of hand labeling became more speedy than last year.

It is practically difficult to pick up the little or complex shape in images, although the process is easier than last year. Therefore, we segmented the image into regions (by colors) in advance with utilizing the segmentation algorithm that is inspired by the paper [23]. In the paper, their segmentation consists of three steps:

- a *Hierarchical and Pyramidal Merging*, initialized from the pixels,
- a *'Video Scan' (or 'Data Flow') Merging*, adapted for the pyramidal region,
- a *Color Merging*, merging step based on a color classification of regions.

Our segmentation algorithm consists of these two previous steps. The algorithm segments the image into regions according to the YUV values. We have only to click a certain region in the segmented image, so that the points in the region are labeled as the selected symbolic color as shown in Fig. 2.2. The algorithm made the hand labeling process easier and faster.

Our learning tool became more useful than last year, and we were able to pickup much training data. However, it takes much learning time, because the learning tool must calculate the influence of all training data over the whole color

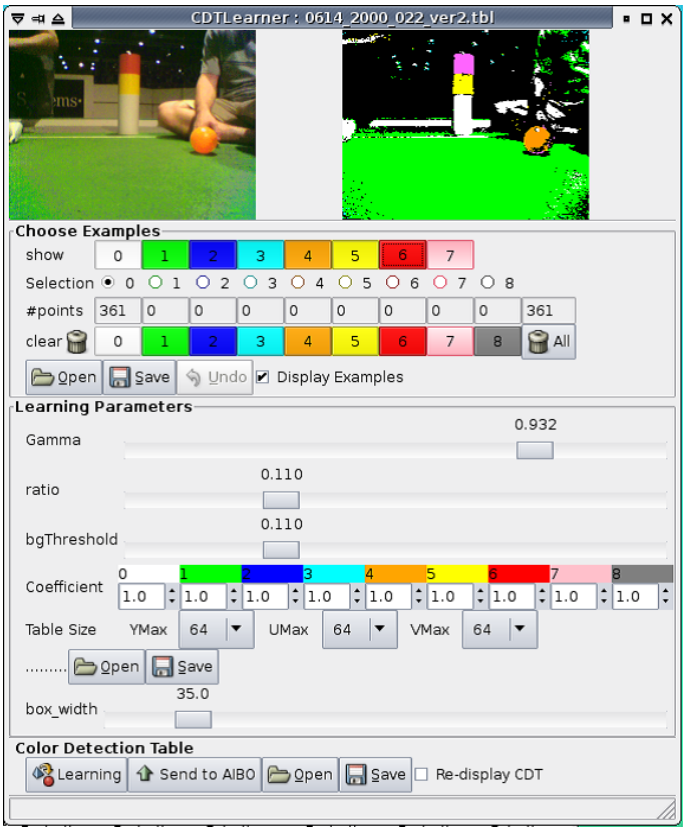


Figure 2.1: Our learning tool.

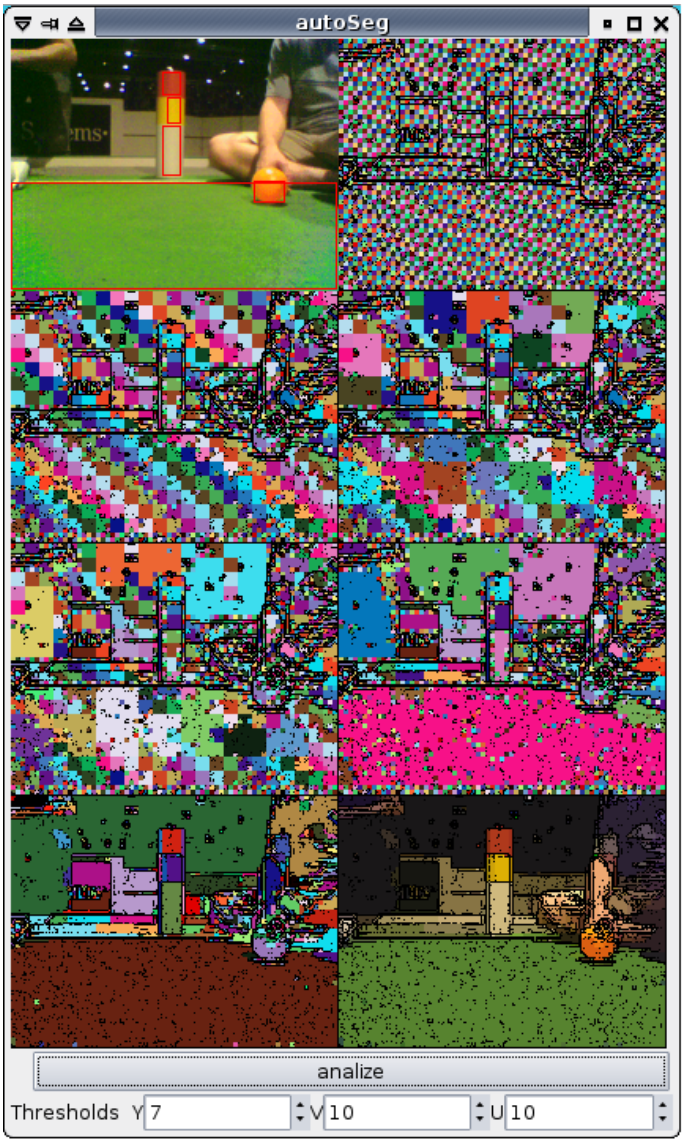


Figure 2.2: Segmentation tool.

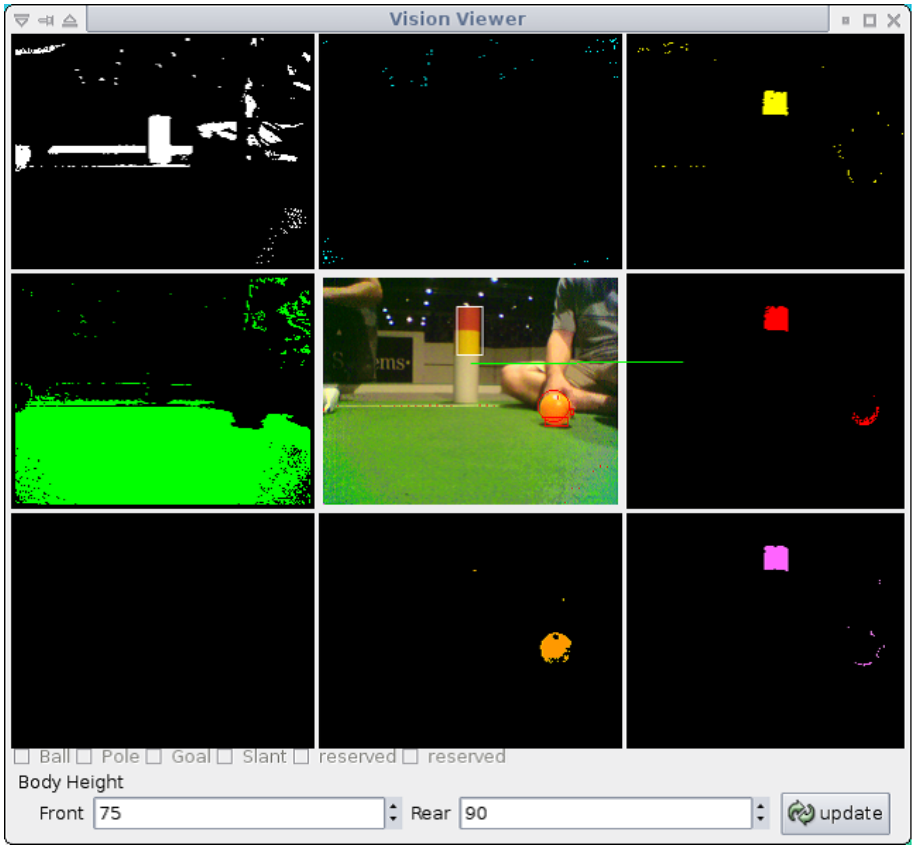


Figure 2.3: The color table segments image.

space. Therefore, we improved the learning tool so that it can calculate the influence of training data only near each point in the whole color space, because the influence is negligible little if the training data are far away from the point.

We improved the learning tool and were able to make more correct color tables. However, the dark orange (e.g. shade of the ball) can be misclassified into red, or the dark yellow (e.g. shade of the yellow goal) can be misclassified into orange. According to the paper [2], this color misclassification is liable to be caused by the one-to-one mapping from one point on the image to one symbolic color. Therefore, we allowed our color tables for mapping from one color to several symbolic colors. As shown in Fig. 2.3, some points are labeled as several symbolic colors.

2.1 Object Recognition

Last year, our object recognition routine caused misidentification occasionally even if the color table was made firmly. As a result, our players sometimes was not able to play a game well. Therefore, we reinforced the routine by adding various

restrictions in this year.

2.1.1 Ball Recognition

Last year, we especially suffered from ball misidentification. For example, red uniforms and yellow-pink boundaries were often mis-recognized as a ball. Thus, we added some restrictions to our ball recognition routine. Our ball recognition routine has two different methods. One is used when the whole of the ball is in the image, and the other is used when the part of it is in the image.

The restrictions of former method is as follows. First, when yellow and pink pixels are detected both above and below ball candidates at the same time, the candidates are ignored because they are inferred not as a ball but as the yellow-pink boundary of a beacon. Second, when red or white pixels are detected more than three times around ball candidates, the candidates are ignored because they are inferred not as a ball but as a red uniform. Third, when green pixels are not detected around ball candidates, the candidates are ignored because they are inferred as a ball not on the field.

The restrictions of latter method is as follows. Last year, a red player's uniform was sometimes misunderstood as the ball which intersects the edges of the image. Therefore, we added a process which counts pixels of the ball candidates which intersect them. Ball's pixels are able to count by computing its area. When pixels are too few, the ball candidates are ignored because they inferred as a red uniform.

2.1.2 Landmark Recognition

Beacon Recognition

As regards beacon recognition, an audience is often mis-recognized as a pole. Therefore, we added the following restrictions.

First, when beacon candidates are too high or too low, the candidates are ignored. Second, when white pixels are not detected below beacon candidates, the candidates are ignored. Third, both a beacon and a diagonally opposite beacon are recognized at the same time, the only nearby beacon is accepted, and the other is rejected.

Goal Recognition

Finally, as regards goal recognition, a straw wall and an audience are mis-recognized as a goal. Therefore, we added the following restrictions.

First, green pixels are not detected below goal candidates, the candidates are ignored. Second, green pixels are detected above goal candidates, the candidates are ignored. In addition, we improved the recognition routine by correlation in the same manner as the beacon recognition.

2.2 Player Recognition

This year, we developed a player recognition routine. The player recognition routine is a part of our vision module.

All players wear a team color (red or blue) uniform. Therefore, we used the uniform for a target of the player recognition. The vision module computes connected components for a specific color image that received from our camera module, and sends team color components' information to the player recognition routine. The recognition routine recognizes each of them to be a player. The red square shown in Fig. 2.4 indicates a part recognized as a player. This routine was used by the Passing Challenge and New Goal Challenge. The details of the routine are as follows.

1. The player recognition routine receives team color components' information from the vision module.
2. The first indication is that the length of the edge of each component is longer than 3 pixels.
3. For each component, the total number of pixels must be over 5 pixels.
4. We calculate a centroid (cx, cy) for each component.
5. We check right and left colors of each component by line scanning.
6. If white pixels are detected at the same time, we recognize the component as a player.

Besides, we developed a visualization tool for our player recognition routine showed in Fig. 2.5. The red arrow in the figure means the direction to an opposite player recognized by the routine.

This routine has been inadequate yet, because it can only output an angle to another player. The following items are the problems for the next year.

- To calculate a more correct estimation of distance from the player
- To change the ending point of the distance estimation from the uniform to an under foot of the player
- To improve the accuracy of the recognition when two or more players are seen in camera view
- To improve the processing speed of this routine

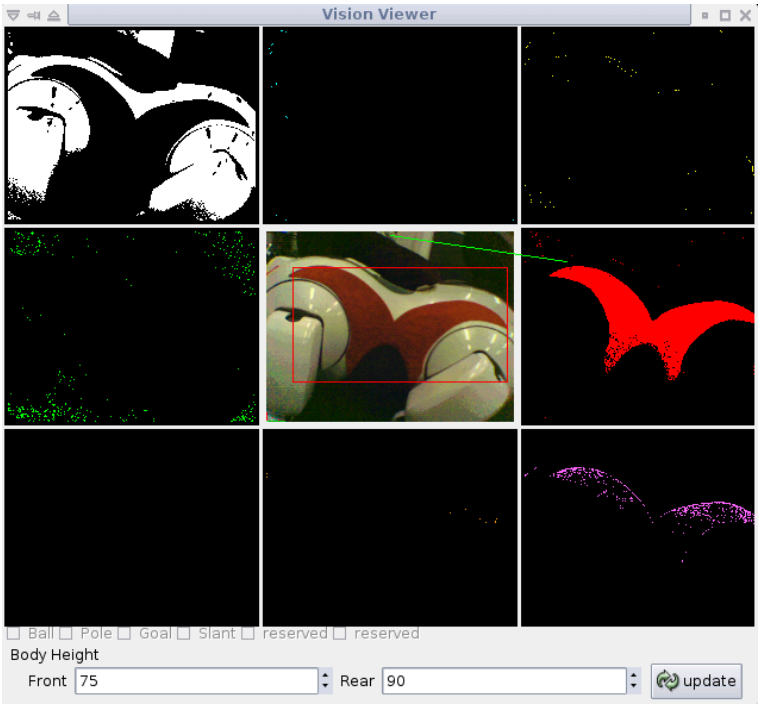


Figure 2.4: Player recognition

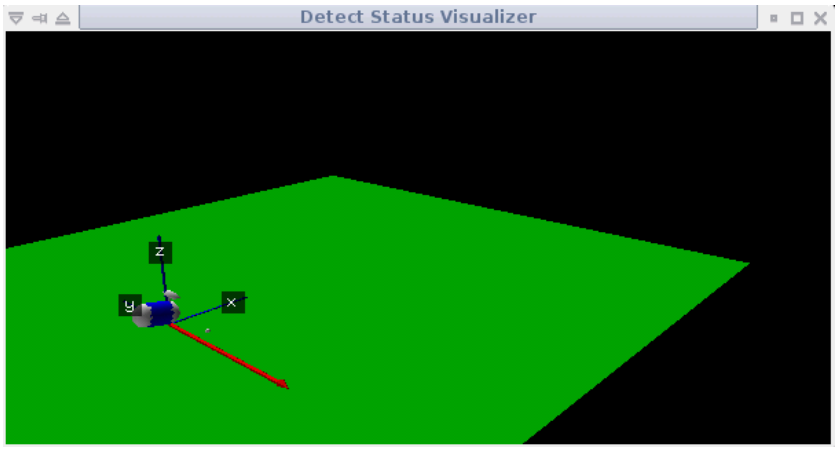


Figure 2.5: Visualization

Chapter 3

Ball Localization

3.1 Introduction

In RoboCup, estimating the position of a ball accurately is one of the most important tasks. Most robots in Middle-size League have omni-directional cameras, and in Small-size League, they use ceiling cameras to recognize objects. In the Four-legged League, however, robots have only low resolution cameras with a narrow field of vision on their noses, while the size of the soccer field is rather large. These facts make it difficult for the robots to estimate the position of the ball correctly, especially when the ball is far away or too near. Moreover, since the shutter speed of the camera is not especially fast, and its frame rate is not very high, tracking a moving object is a demanding task for the robots. In order to tackle these problems, several techniques have been proposed [22, 20]. Kalman filter method [13] is one of the most popular techniques for estimating the position of objects in a noisy environment. It is also used to track a moving ball [4]. Methods using particle filters have been applied for tracking objects [10], and especially in Four-legged League, the method using Rao-Blackwellised particle filters has had great success[20].

In this chapter, we consider how to estimate the trajectory of the moving ball, as well as the position of the static ball, based on the Monte-Carlo localization method [3]. Monte-Carlo localization has usually been used for self-localization in the Four-legged League. We extend this to handle the localization of the ball. We have already mentioned a primitive method based on this idea in [11], where we had ignored the velocity of the ball. We propose three extended methods to deal with the velocity of the ball and report some experimental results.

3.2 Ball Recognition from the Image Data

As we have already mentioned in the previous section, the camera of the robot in Four-legged league is equipped on its nose. The field of vision is narrow (56.9° for horizontal and 45.2° for vertical), and the resolution is also low (412×320 at the maximum). Moreover, stereo vision is usually impossible, although some attempts

Figure 3.1: Cases of Ball Recognition. Dots on the edge show vertexes of a inscribed triangle. The diameter of the ball is calculated by using these vertexes.

to emulate it by using plural robots have been reported [14, 26]. Therefore, first of all, an accurate recognition of the ball from a single image captured by the camera is indispensable in estimating the position of the ball accurately. In particular, the estimation of the distance to the ball from the robot is very critical in establishing a stable method for estimating the position of the (possibly moving) ball.

A naive algorithm which counts the orange pixels in the image and uses it to estimate the distance does not work well, since the ball is often hidden by other robots, and the ball may be in the corner of the robot's view as shown in Figure 3.1. In addition, the projection of the line of sight to the ground is used for calculating the distance to the object, but it depends on the pose of the robot and is affected by the locomotion of the robot. In the Four-legged League, the robot's sight is very shaky; therefore, we do not use this method either. In this section, we will show our algorithm ability to recognize the ball and estimate the position relative to the robot from a single image. It will become the base for estimations from multiple images, as described in the following sections.

In the image, the biggest component colored by orange and satisfying the following heuristic conditions is recognized as the ball. (1) The edge length of the component has to be more than 10 pixels, which helps exclude components too small. (2) The ratio of the orange pixels to the area of the bounding box must exceed 40%. (3) If the component touches the edge of the image, the length of the longer side of the bounding box must be over 20 pixels.

We use the diameter of the ball to estimate the distance to it. However, the ball is often hidden by other robots partially, and when the robot approaches to the ball, only a part of the ball is visible at the corner of the camera view. Thus the size of the bounding box and the total number of pixels are not enough to estimate the distance accurately.

Figure 3.1 shows two cases of diameter estimation. When the ball is inside the view completely (left image), we regard the length of longer side of the bounding box as the diameter of the ball. When the bounding box touches to the edges of the image (right image), we use three points of the components, since any three points of the edge of a circle uniquely determine the center and the diameter of it.

3.3 Ball Localization using Monte-Carlo Method

In this section, we propose a technique which estimates the position and velocity of a moving ball based on the Monte-Carlo localization. This technique was introduced for the self-localization in [7] and utilized in [21]. The aim is to calculate the accurate position and velocity of the ball from a series of input images. In principle, we use differences of positions between two frames. However, these data may contain some errors. The Monte-Carlo method absorbs these errors so that the estimation of the velocity becomes more accurate. In this method, instead of describing the probability density function itself, it is represented as a set of *samples* that are randomly drawn from it. This density representation is updated every time based on the information from the sensors.

We consider the following three variations of the method, in order to estimate both the position and velocity of the moving ball. (1) Each sample holds both the position and velocity, but is updated according to only the information of the position. (2) Each sample holds both the position and velocity, and is updated according to the information of both the position and velocity. (3) Two kinds of samples are considered: one for the position, and the other for the velocity, which are updated separately. For evaluating the effectiveness of our ball tracking system, we experimented in both simulated and real-world environments. The details of the algorithms and experimental results are shown below.

3.3.1 Ball Monte-Carlo localization updated by only position information

First, we introduce a simple extension to our ball localization technique in [11]. In this method, a sample is a triple $a_i = \langle \vec{p}_i, \vec{v}_i, s_i \rangle$, where $\vec{p}_i$ is a position of the ball, $\vec{v}_i$ is a velocity of it, and s_i is a score ($1 \leq i \leq n$) which represents how $\vec{p}_i$ and $\vec{v}_i$ fit to observations. Fig 3.2 shows the update procedure.

When the robot recognizes a ball in the image, each score s_i is updated in step $6 \sim 10$, depending on the distance of $\vec{p}_i$ and the observed position $\vec{p}_o$. When the ball is not found, let $\vec{p}_o = \epsilon$. The constant $\tau > 0$ defines the *neighborhood* of the sample. Moreover, the constants *maxup* and *maxdown* control the *conservativeness* of the update. If the ratio *maxup*/*maxdown* is small, response to the rapid change of ball's position becomes fast, but influence of errors become strong. The threshold value t controls the effect of misunderstandings. The function *randomize* resets p_i and v_i to random values and $s_i = 0$. The function *random*() returns a real number randomly chosen between 0 and 1. The return values $\vec{p}_e$ and $\vec{v}_e$ are weighted mean of samples whose scores are at least t.

At first, we show a simulation results on this method. The simulation is performed as follows. We choose 20 positions from $(800, 1300)$ to $(-800, 1300)$ at regular intervals as *observed positions*, which emulates that a robot stands at the center of the field without any movement and watches the ball rolling from left to right. Since in practice, the distance estimation to the ball is not very accurate

Algorithm BallMoteCarloLocalizationUpdatedByOnlyPositionInformation
Input. A set of samples $Q = \{\langle \vec{p}_i, \vec{v}_i, s_i \rangle\}$ and observed ball position $\vec{p}_o$
Output. estimated ball position $\vec{p}_e$ and velocity $\vec{v}_e$.

1 **for** $i := 1$ **to** n **do begin**
2 $\vec{p}_i := \vec{p}_i + \vec{v}_i$;
3 **if** $\vec{p}_i$ is out of the field **then**
4 *randomize*($\langle p_i, v_i, s_i \rangle$)
5 **end**;
6 **if** $\vec{p}_o \neq \epsilon$ **then**
7 **for** $i := 1$ **to** n **do begin**
8 $score := exp(-\tau|\vec{p}_i - \vec{p}_o|)$;
9 $s_i := \max(s_i - maxdown, \min(s_i + maxup, score))$
10 **end**
11 $avgScore := \frac{1}{n} \sum_{i=1}^{n} s_i$;
12 $\vec{p}_e := \vec{0}$; $\vec{v}_e := \vec{0}$; $w := 0$;
13 **for** $i := 1$ **to** n **do begin**
14 **if** $s_i < avgScore \cdot random()$ **then**
15 *randomize*($\langle p_i, v_i, s_i \rangle$)
16 **else if** $s_i > t$ **then begin**
17 $\vec{p}_e := \vec{p}_e + s_i \vec{p}_i$;
18 $\vec{v}_e := \vec{v}_e + s_i \vec{v}_i$;
19 $w := w + s_i$
20 **end**;
21 **end**;
22 $\vec{p}_e := \vec{p}_e / w$; $\vec{v}_e := \vec{v}_e / w$;
23 **output** $\vec{p}_e$, $\vec{v}_e$

Figure 3.2: Procedure Ball Monte-Carlo Localization updated by only position information

compared to the direction estimation, we added strong noise to y-direction and weak noise to x-direction of the observed positions. These positions are shown as diamond-shaped points in Fig. 3.3 and Fig. 3.4. We examined the above method and a simple Kalman filter method to predict the positions of the ball from these noisy data. The lines in Fig. 3.3 and 3.4 shows the trajectories of them for 40 frames (in the first 20 frames, the observed positions are given, and in the last 20 frames, no observed data is given). After some trials, we set the parameters as follows. The number of samples $n = 500$, $maxup = 0.1$, $maxdown = 0.05$, $t = 0.15$, and $\tau = 0.008$. Compared to Kalman filter, our method failed to predict the trajectory of the positions after the ball is out of the view, because the velocity of the samples did not converge well.

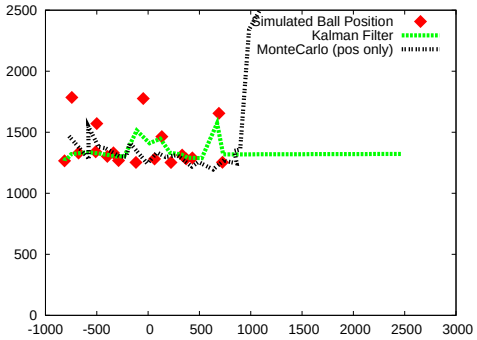


Figure 3.3: The result of the simulation with Kalman filter and MonteCarlo(pos only)

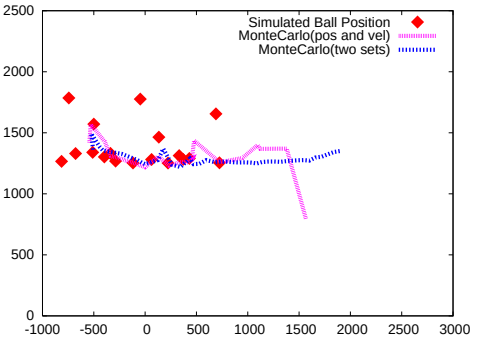


Figure 3.4: The result of the simulation with MonteCarlo(pos and vel) and MonteCarlo(two sets)

3.3.2 Ball Monte-Carlo localization updated by position and velocity information

In the previous method, each sample holds its own velocity, and in principal, good sample holding good position and velocity would get high score. However in the previous experiment, the velocity did not converged well. Therefore, we tried to reflect the velocity explicitly to the score, intended to accelerate the convergence. In the second method, we added the procedure shown in Fig. 3.5 to the previous method, between Step 10 and Step 11 in Fig. 3.2. Since the *observed velocity* $\vec{v}_o$ is not given explicitly, it is substituted by $(\vec{p}_o - \vec{p}_{last})/dt$, where $\vec{p}_{last}$ is the last position of the ball and dt is the difference between these two frame numbers.

The line *MonteCarlo(pos and vel)* in Fig. 3.4 shows the result, with the parameters $\tau_v = 0.08$, $maxup_v = 1.0$, $maxdown_v = 0.1$. Unfortunately, it also failed to predict the trajectory because the converge rate was not improved well.

for $i := 1$ **to** n **do begin**
  $score := exp(-\tau_v|\vec{v}_i - \vec{v}_o|)$;
  $s_i := \max(s_i - maxdown_v, \min(s_i + maxup_v, score))$
end

Figure 3.5: Additional procedure to update the score reflecting the velocity explicitly in the second method.

3.3.3 Ball Monte-Carlo localization with two sets of samples

In the third method, we took another approach in order to reflect the goodness of the velocity to the score. The idea is to split the samples into two categories, one for the positions $\langle \vec{p}_i, s_i \rangle$, and the other for the velocities $\langle \vec{v}_i, s_i \rangle$. The score of $\vec{p}$ is updated in step $8 \sim 11$ of *PositionUpdate* while that of $\vec{v}$ in step $2 \sim 5$ of *VelocityUpdate* independently, in Fig. 3.6.

The line *MonteCarlo(two sets)* in Fig. 3.4 shows the results. The estimated positions fits the pathway of the rolling ball, and the predicted trajectory when ball was out of sight is also as we expected. The effectiveness of it is comparable to the Kalman filter method.

3.3.4 On the number of samples

We verified the relationship between the accuracy of the predictions and the number of samples. Fig. 3.7 shows some of these experiments for the proposed three methods. The x-axis is the number of samples ranging 10 to 1000, and the y-axis is the average error of five trials, which measure the difference between the real position and the predicted position. In the left figure, the situation was almost the same as the previous experiments. In the right figure, we put a small obstacle in front of the robot which hides the ball in the center of the view. Because of this obstacle, some of observed positions were missing and the average errors increased. In general, the error decreases as the number of samples increases for all of these three methods. Among them, the third method which separates the samples into two categories performed the best. When the number of samples exceeds 150, the accuracy did not changed in practice.

Remind that the data on observed positions, which are given as input, themselves contained some errors. For example, the average error of the observed positions was 52.6 mm in the experiment without obstacle. However, the average error of the third method was less than 40 mm when we took at least 150 samples, which verified that the proposed methods succeeded to stabilize the predicted ball positions.

3.3.5 Real-world Experiments

We also performed many experiments in the real-world environments, at which we used the real robot in the soccer field for RoboCup competitions. We show

Algorithm BallMoteCarloLocalizationWithTwoSets
Input. Two sets of samples $POS = \{\langle \vec{p}_i, s_i \rangle\}$ and $VEL = \{\langle \vec{v}_i, s_i \rangle\}$, observed ball position $\vec{p}_o$, calculated ball velocity $\vec{v}_o$.
Output. estimated ball position $\vec{p}_e$ and velocity $\vec{v}_e$.

PositionUpdate$(POS, VEL, \vec{p}_o, \vec{v}_o)$

```
1   v_e := VelocityUpdate(VEL, p_o, v_o);
2   for i := 1 to n do begin
3     p_i := p_i + v_e;
4     if p_i is out of the field then
5       randomize(⟨p_i, s_i⟩)
6   end;
7   if p_o ≠ ε then
8     for i := 1 to n do begin
9       score := exp(−τ_p|p_i − p_o|);
10      s_i := max(s_i − maxdown_p,
                   min(s_i + maxup_p, score))
11    end;
12  p_e = 0;    w = 0;
13  avgScore := (1/n) Σ_{i=1}^{n} s_i;
14  for i := 1 to n do begin
15    if s_i < avgScore · random() then
16      randomize(⟨p_i, s_i⟩)
17    else if s_i > t_p then begin
18      p_e := p_e + s_i p_i;
19      w := w + s_i
20    end;
21  end;
22  p_e := p_e/w;
23  output p_e, v_e
```

VelocityUpdate$(VEL, \vec{p}_o, \vec{v}_o)$

```
1   if p_o ≠ ε then
2     for i := 1 to m do begin
3       score_new := exp(−τ_v|v_i − v_o|);
4       s_i := max(s_i − maxdown_v,
                   min(s_i + maxup_v, score))
5     end;
6   v_e = 0;    w = 0;
7   avgScore := (1/m) Σ_{i=1}^{m} s_i;
8   for i := 1 to m do begin
9     if s_i < avgScore · random() then
10      randomize(⟨v_i, s_i⟩)
11    else if s_i > t_v then begin
12      v_e := v_e + s_i v_i;
13      w := w + s_i
14    end;
15  end;
16  v_e := v_e/w;
17  output v_e
```

Figure 3.6: Procedure Ball Monte-Carlo Localization with two sets of samples

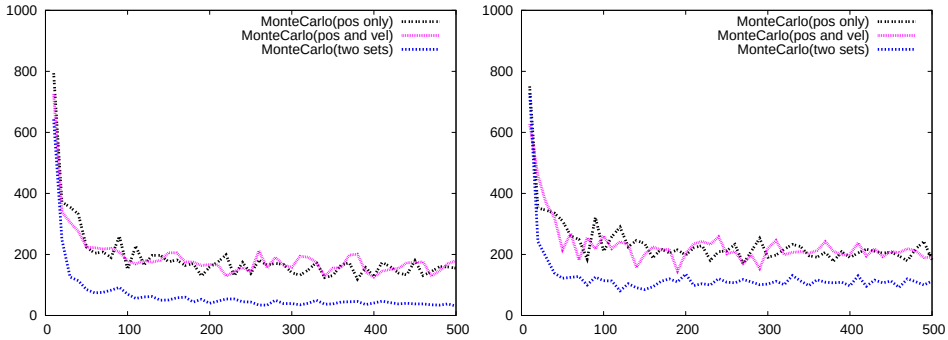


Figure 3.7: Relationship between the number of samples and accuracy, in two situations, without obstacle (left) and with an obstacle (right).

some of them in Fig. 3.8, where we compared the third method which we proposed with the Kalman filter method. The purpose was to evaluate the robustness of the methods against the obstacle and the change directions of the ball movement. In the left figure, the ball rolled from right to left while a small obstacle in front of the robot hides for some moments. In the right figure, the ball was kicked at $(400, 900)$ and went left, then it is rebounded twice at $(-220, 1400)$ and $(-50, 500)$, and disappeared from the view to the right. Mesh parts in the figures illustrates the visible area of the robot. From these experiments, we verified that the proposed method is robust against the frequent change of the directions, which is often the case in real plays.

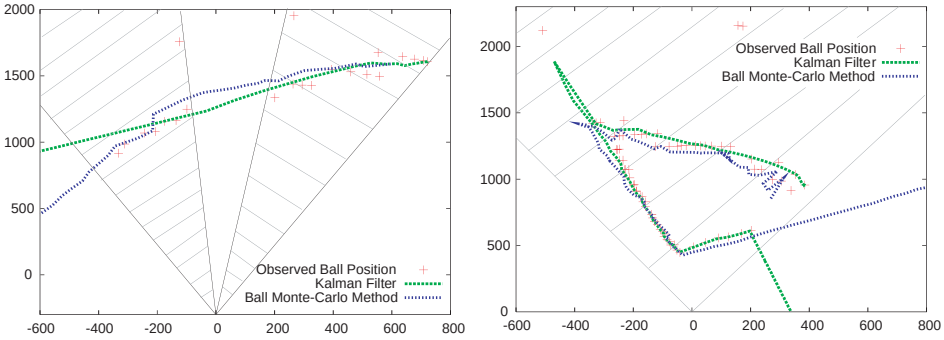


Figure 3.8: Real world experiments. In the left situation, the ball rolled from right to left behind a small obstacle. In the right situation, the ball started at $(400, 900)$ and rebounded twice at $(-220, 1400)$ and $(-50, 500)$, and disappeared from the view to the right.

3.4 Conclusions

We proposed some applications of Monte-Carlo localization technique to track a moving ball in noisy environment, and showed some experimental results. We tried three variations, depending on how to treat the velocity of the ball movement. The first two methods did not work well as we expected, where each sample has its own velocity value together with the the position value. The third method, which we treated positions and velocities separately, worked the best. We think the reason as follows. If we treat positions and velocities together, the search space has four dimensions. On the other hand, if we treat them separately, the search space is divided into two spaces, each of them has two dimensions. The latter would be easy to converge to correct values. The converge ratio is quite critical in RoboCup, because the situation changes very quickly in real games.

We compared the proposed method with Kalman filter method, which is very popular. The experiments were not very comprehensive: for example, the observing robot did not move. Nevertheless, we verified that the proposed method is attractive especially when the ball rebounds frequently in real situations.

In RoboCup domain, various techniques to improve the ball tracking ability are proposed. For example, cooperative estimation with other robots is proposed in [5, 14]. we plan to integrate these idea into our method, and perform more comprehensive experiments in near future.

Chapter 4

Learning of Ball Trapping

4.1 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 chapter, 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 [24, 9] 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 to-

day [8, 15, 18, 28]. 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 [1] studied the learning of ball kicking skills, which is an important skill directly related to scoring points. Zagal and Solar [29] 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 [6] 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.

4.2 Preliminary

4.2.1 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. 4.1 (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 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. 4.1 (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

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

Figure 4.1: The motion to actually trap the ball (a), and the motion to judge if it succeeded in trapping the ball (b).

with.

4.2.2 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. 4.2. 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 4.7.

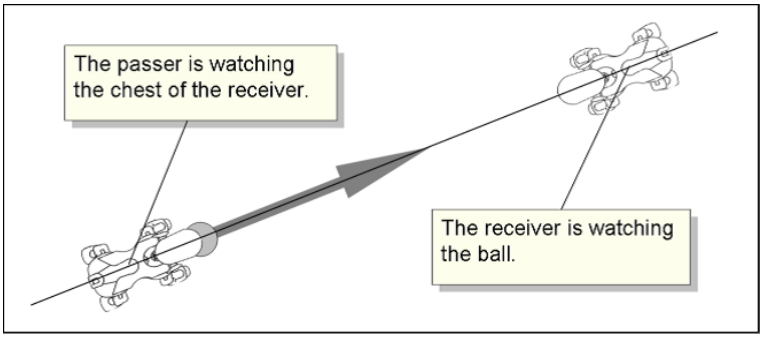


Figure 4.2: One-dimensional model of ball trapping problem.

Figure 4.3: Training equipment for learning ball trapping skills.

4.3 Training Equipment

The equipment we prepared for learning ball trapping skills in one-dimensional is fairly simple. As shown in Fig. 4.3, 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.

4.4 Learning Method

Fidelman and Stone [6] 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 chapter, we apply reinforcement learning algorithms [27]. 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 4.2.2, 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 4.2.1. 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. 4.4 shows the autonomous learning algorithm used in our research. It is a combination of the episodic SMDP Sarsa(λ) 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 [24].

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 $\overrightarrow{\theta}$ 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 $\overrightarrow{\theta}$ for each value of F_a. The policy ϵ*-greedy* selects a random action with probability ϵ, and otherwise, it selects the action with the maximum Q-value. We set $\epsilon = 0.01$. Moreover, $\overrightarrow{e}$ is an *eligibility trace*, which stores the credit that past action choices should receive for current rewards. λ 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* $\gamma = 1.0$.

4.5 Experiments

4.5.1 Training Using One Robot

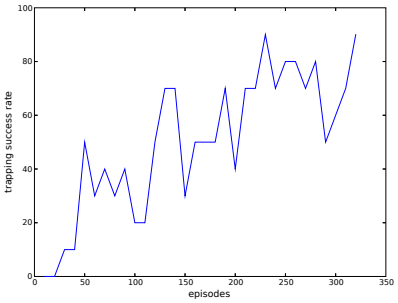
We first experimented by using one robot along with the training equipment that was illustrated in Section 4.3. The robot could train in solitude and learn ball trapping skills on its own.

Fig. 4.5(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.

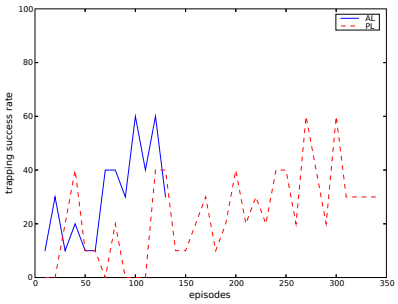
Fig. 4.6 shows the result of each episode by plotting a circle if it was successful, a cross if it failed in spite of trying to trap, and a triangle if it failed because of doing nothing. From the 1st episode to the 50th episode, the robots simply tried to trap the ball while it was moving with various 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

```
1  while still not acquiring trapping skills do
2      go get the ball and move to a random position with the ball;
3      kick the ball toward the slope;
4      s ← a state observed in the real environment;
5      forall a ∈ 𝒜(s) do
6          F_a ← set of tiles for a, s;
7          Q_a ← Σ_{i∈F_a} θ(i);
8      end
9      lastAction ← an optimal action selected by ε-greedy;
10     e⃗ ← 0;
11     forall i ∈ F_lastAction do e(i) ← 1;
12     reward ← 0;
13     while reward = 0 do
14         do lastAction;
15         if lastAction = trap then
16             if the ball is held then reward ← 1;
17             else reward ← −1;
18         else
19             if collision occurs then reward ← −1;
20             else reward ← 0;
21         end
22         δ ← reward − Q_lastAction;
23         s ← a state observed in the real environment;
24         forall a ∈ 𝒜(s) do
25             F_a ← set of tiles for a, s;
26             Q_a ← Σ_{i∈F_a} θ(i);
27         end
28         lastAction ← an optimal action selected by ε-greedy;
29         δ ← δ + Q_lastAction;
30         θ⃗ ← θ⃗ + αδe⃗;
31         Q_lastAction ← Σ_{i∈F_lastAction} θ(i);
32         e⃗ ← λe⃗;
33         if player acting in state s then
34             forall a ∈ 𝒜(s) s.t. a ≠ lastAction do
35                 forall i ∈ F_a do e(i) ← 0;
36             end
37             forall i ∈ F_lastAction do e(i) ← 1;
38         end
39     end
40     δ ← reward − Q_lastAction;
41     θ⃗ ← θ⃗ + αδe⃗;
42 end
```

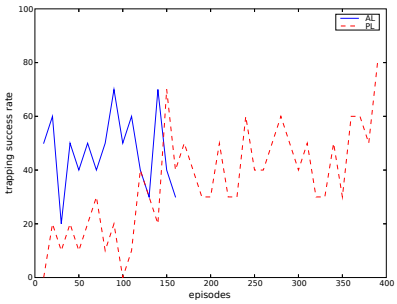
Figure 4.4: Algorithm of our autonomous learning (based on keepaway learner [24]).



(a) one robot

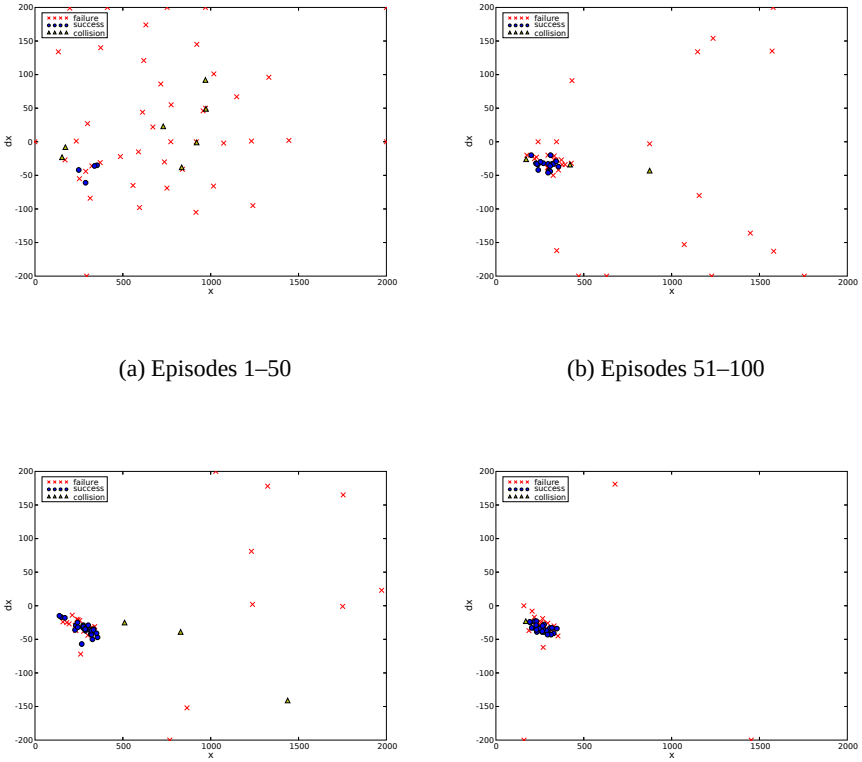


(b) two robots



(c) two robots with communication

Figure 4.5: Results of three experiments.



(a) Episodes 1–50

(b) Episodes 51–100

(c) Episodes 101–150

(d) Episodes 151–200

Figure 4.6: 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.

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.

4.5.2 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 search out nor approach the ball if the trapping failed. Only AL does

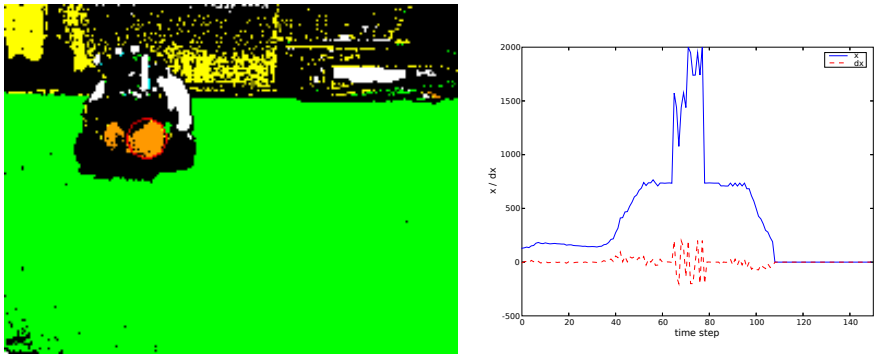


Figure 4.7: The left figure shows how our vision system recognizes a ball when the other robot holds it. The ball looks to be smaller than it is, because a part of it is hidden by the partner and its shadow, resulting in an estimated distance to the ball that is further away than it really is. The right 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.

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. 4.7. 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 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. 4.5(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.

4.5.3 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.

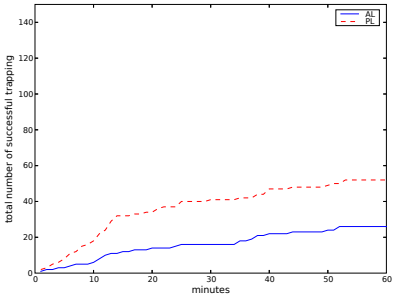
First, we made the robots share their experiences with each other, as in [19]. However, if they continuously communicated with each other, they could not do anything else, because the excessive processing would interrupt the input of proper states from the real-time environment. Therefore, we made the 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 $\vec{\theta}$ 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 4.5.2 using two robots which can communicate with each other. Fig. 4.5(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.

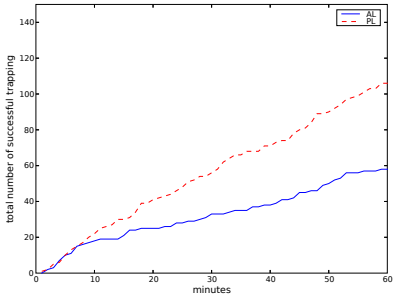
4.6 Discussion

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. 4.8 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. 4.8(a) and Fig. 4.8(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. 4.8(c) and Fig. 4.8(d) shows the learning process with and without communication, respectively, after pre-learning for 60 minutes in solitude.

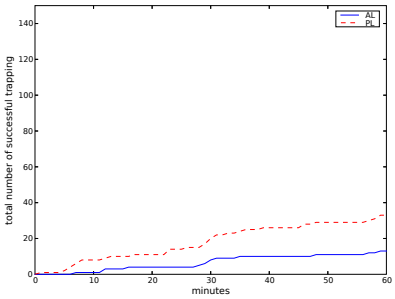
Comparing (a) and (c) with (b) and (d) has us conclude that allowing AL and



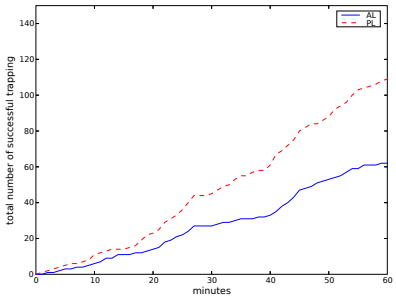
(a) without communication after pre-learning by using two robots



(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

Figure 4.8: Total numbers of successful trappings with respect to the elapsed time.

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 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.

4.7 Conclusions and Future Work

In this chapter, 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 [25], 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 [24] techniques for use in the four-legged robot league.

Chapter 5

Strategy System

This year, we changed the description method of our high-level strategies to a simple if-then rule method from a state transition method. The state transition method allows us to quickly develop step-by-step actions, e.g., actions which perform a certain action after achieving another certain action. This method, however, results in annoying debugging work when we create huge programs, because we must carefully make sure that all transitions in all states are valid (otherwise, robots may stop moving because of an infinite loop transition or a transition to an unknown state). On the other hand, the if-then rule method allows us to create readable and brief program codes which we can debug easily, although we must write more complex program codes for such step-by-step actions. We intend to utilize machine learning techniques for acquiring more complex rules as sophisticated strategies next year.

We also changed our behavior system, by which our low-level strategies were created, for new high-level strategies. Our behavior system still supports the state transition method, because low-level strategies mostly include step-by-step actions.

5.1 Behavior System

The new behavior system is based on our behavior system of last year. The difference from the old one is to utilize two types of behaviors. One is a normal behavior, and the other is a non-stop behavior. The normal behaviors can be exchanged anytime, while the non-stop behaviors can never be exchanged until they are terminated. For example, the normal behaviors include approaching, positioning, dribbling, and so on; The non-stop behaviors include shooting, guarding, catching, and so on, which must be continued even if the situation is changed (e.g., the ball is lost) during executing the behaviors.

The following code is an example of behavior creation in our system. The function “init” is called only once at the beginning. The function “setState” changes the behavior’s state, which is also a function name that called by the system every decision-making time. This example prints the string “action01” only once, i.e. the function “init”, “action01”, “finish”, “finish”, ... are called during executing

the behavior “foo”, in that order.

```
foo = Behavior("foo") -- or NonstopBehavior("foo")

function foo.init()
  foo.setState("action01")
end

function foo.action01()
  print("action01")
  foo.setState("finish")
end

function foo.finish()
end
```

Moreover, we established a combo behavior mechanism that can combine several behaviors. For example, we can create a behavior shooting a ball after catching it by combining a shooting behavior and a catching behavior. The following code is an example of combining two behaviors. The combo behavior “comboFooBar” means the behavior “bar” is executed right after the behavior “foo” is terminated.

```
foo = Behavior("foo")
...
bar = Behavior("bar")
...
comboFooBar = ComboBehavior(foo, bar)
```

Chapter 6

Strategies for Jolly Pochie 2006

6.1 Attacker

This year, we used two types of attacker scripts. One is "*AtFG*" we call. It designed for games. The other is "*AtMAX*". It designed for the penalty shoot-out and the new goal challenge. The difference between these two scripts is the way of shooting and approaching. Our attacker has some various shots in games. It chooses one of them depending on the situation. However it shoots only one way in the penalty shoot-out. The shoot is very strong and carries the ball far. In addition the player approaches the ball more carefully in penalty shoot-out than in games. In penalty shoot-out, this is because the attacker needs to catch the ball more surely.

Our attacker was designed with if-then rule. Last year it was designed based on a state transition method. This method, however, often causes bugs in script codes. This year our goalie script was written with a if-then rule method. The if-then rule is very simple and has less incidence of bugs. We describe these details in the following sections.

The attacker has the following five processes. We named them as "*search*", "*approach*", "*shoot*", "*support*", and "*localize*". Each of them is used depending on the situation.

6.1.1 Search

The attacker searches the ball in the "*search*" process. When the attacker can not see the ball, the "*search*" process is called to find the ball. This process consists of the following behaviors.

- "*SearchWalk*": The attacker searches the ball with walking from one side of the field to the other.
- "*SearchDown*": The attacker searches the ball under its head by swinging its head.
- "*SearchTurn*": The attacker searches the ball with turning right or left.

We improved the method of searching the ball. When the attacker lost the ball, it looked near and far at first in the last year. If it was not able to find the ball even though this process had been over, it turned to the rotative direction where the attacker had seen the ball last time. This year the attacker turned after swinging its own head only once.

6.1.2 Approach

The attacker approaches the ball enough to shoot or catch it in the "*approach*" process. When it watches the ball, the "*approach*" process is called.

6.1.3 Shoot

The attacker tries to score in the "*shoot*" process. When the attacker is near the ball enough to shoot or catch it, this process is called.

If the attacker can catch it easily, it tries to do. After catching it, the attacker turns to the direction of the opponent goal. If it can see the opponent goal or the regulation time for continuous catching is over, the attacker shoots it.

If the attacker can not catch easily, it tries to shoot it. The attacker has various shots, and it chooses the best shot for scoring.

6.1.4 Support

We built a communication system for supporting the other teammates. The system allows the players to communicate with each other. They can tell each other some information: their own position, the relative ball position, and the sending time.

We developed two strategies, "*active*" and "*passive*", and made our attackers switch their strategy. In case that any other teammate is not chasing the ball, the attacker plays with the "*active*" strategy. It allows the attacker to use the "*search*", "*approach*" and "*shoot*" processes. Otherwise, that is, in case that other teammate is the nearest to the ball and chasing it, the attacker plays with the "*passive*" strategy. It allows the attacker to use only the "*search*" and "*support*" processes. The attacker can not call the "*approach*" process so that it does not clash with other teammates. This strategy system allows only one attacker to approach the ball. However our defender and goalie do not use the system. This is because they must clear the ball even if they have the risk of clashing other teammates.

When the "*support*" process is called, the attacker does not approach and moves to a supporting position which is between the ball and the opponent goal. The attacker stays in the supporting position until other player moves the ball.

6.1.5 Localize

This year, our attacker is always localizing own position in games, because stopping for the localization wastes its playing time. However, some clashes still often bring the attacker misunderstanding of its own position localization, and it can not

estimate its own position correctly when it is in the "*penalty*" state. In that case, the "*localize*" process is called expressly. When it is called, the attacker stops walking and estimates its own position by looking around carefully.

6.1.6 Problems for next year

We have the following problems that we try to solve next year.

The accuracy of catching and shooting

This year, when the player found the ball, the player approached, caught, and shot it. However, the player often failed to catch the ball, and was not able to shoot it in important occasions. Therefore, we need to improve the catching motion and the shooting motion. As a strategy, we need to adjust the condition of switching the "*shoot*" process and the "*approach*" process.

The avoidance of the penalties of illegal defender

As to illegal defender, we designed the script so that the defender should not enter the own penalty area. However, we did not designed this script for the attacker, because we had thought that the attacker rarely entered in its own penalty area. As a result, the attacker often entered the own penalty area and was received the penalty. Therefore, we need to apply this script to the attacker next year.

The improvement of supporting each other

We need to improve the supporting system. This year, the player switched the "*active*" and "*passive*" strategies depending on the distance of the ball. In the "*passive*" strategy, the player only stays in the place. Therefore, the player needs to do more sophisticated action, e.g., it continues to move dynamically to turn any situation to its advantage.

6.2 Defender

The strategy of our defender is almost the same as that of the attacker. However, the defender should be more defensive than attackers. We devised the strategy of the defender. The detail is as follows.

First, the defender does not go out over a half line. Second, when the ball is an opposite side, the defender stays at its own position. Third, when the ball comes into the defender's face, either the "*guard*" or the "*clear*" process is called depending on the situation.

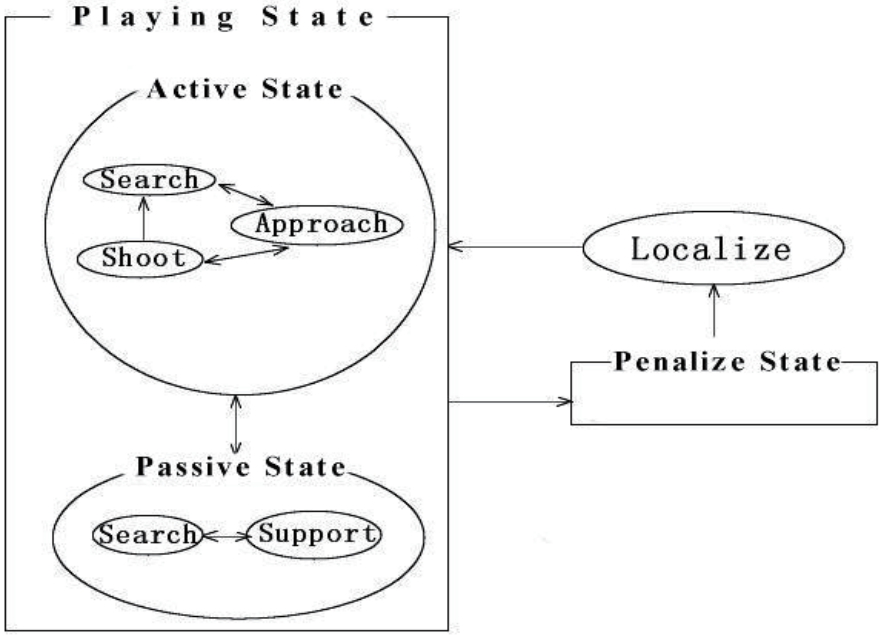


Figure 6.1: The strategy of Attacker

6.3 Goalie

Our goalie's strategy is separated into four processes: "*search*", "*position*", "*guard*", and "*clear*". First, our goalie is positioned at the center of its own goal in the "*position*" process, because it is the most important for the goalie to be in front of its own goal. Second, the "*search*" process is called, if the goalie does not see the ball. The goalie must recognize the current position of the ball to respond to the movement of it. Next, the goalie saves in the "*guard*" process. An opponent would shoot the ball at the chink of the goal. Therefore, the goalie can not defend its own goal only by standing in front of our goal. The goalie must move left or right to fill up the chink and defend its own goal by opening the legs. Finally, the goalie approaches the ball and kicks it out in the "*clear*" process. In case there is the ball inside the penalty area, the goalie must clear it courageously. This year the "*guard*" and "*clear*" processes are especially improved . Each process is described in detail in the following.

6.3.1 Position

The "*position*" process (cf. Fig. 6.2a) is the most important for the goalie. If the goalie is not in front of its own goal at a crucial moment (e.g. when an opponent shoots the ball to our goal), it will be equal to giving an opposing team one point.

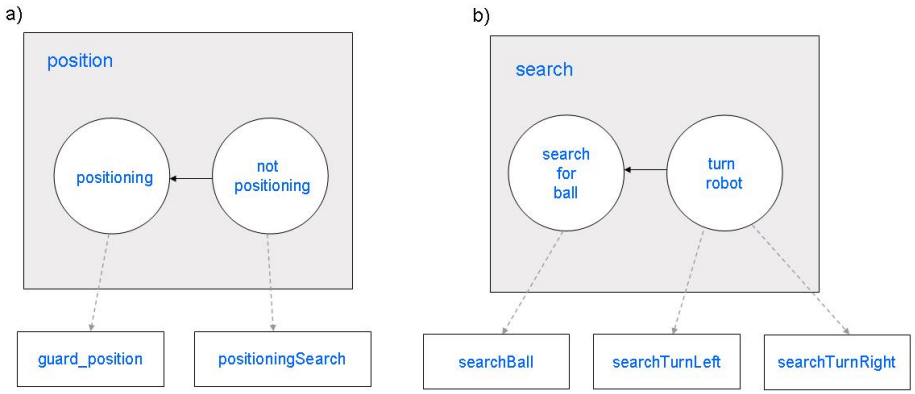


Figure 6.2: The circles are situations and the squares are behaviors. a) The situations and behaviors in the "*position*" process. b)The situations and behaviors in the "*search*" process .

When the goalie is not in a right position, this situation is called "*not-positioning*". In the situation, the goalie with the behavior "*positioningSearch*" moves in front of its own goal and does not collide with one of the goal posts. If the goalie finds the ball or is near it, the goalie continues to move in front of its own goal. This is because it is the most important for the goalie to be in front of its own goal. When the goalie is almost at the right position, this situation is called "*positioning*". In the situation, the goalie walks to a more suitable place (e.g. when there is the ball on the left side, the goalie moves to the left a little.) depending on the position of the ball with the behavior "*guard_position*". When the goalie does not see the ball, the "*search*" process is called. When the ball is near enough to approach it, either the "*guard*" or "*clear*" process is called in the same manner as the "*search*" process.

6.3.2 Search

The "*search*" process (cf. Fig. 6.2b) is called when the goalie is in its own position and does not see the ball. In a situation "*turn-robot*" the goalie searches a direction which it had seen the ball last. In the situation, the goalie turns to the direction with the behavior "*SearchTurnRight*" or "*SearchTurnLeft*". In a situation "*search-for-ball*" the goalie stays in front of its own goal and searches the ball with swinging its head. The "*search*" process is terminated when the goalie finds the ball. After that, in the case that the ball is near to the goalie, either the "*guard*" or the "*clear*" process is called. In the other case, the "*position*" process is called.

6.3.3 Guard

The "*guard*" process (cf. Fig. 6.3a) is called only when the goalie in front of the goal is watching the ball. Last year, although the ball moves slowly, the goalie

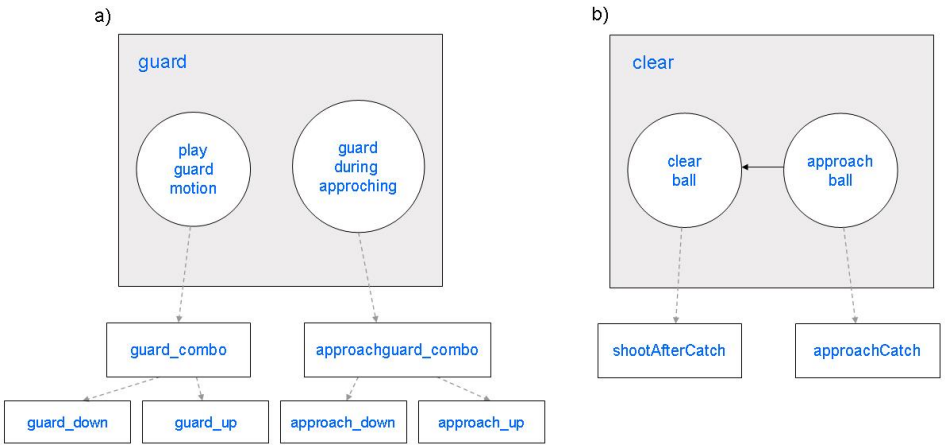


Figure 6.3: a) The situations and behaviors in the "*guard*" process. b) The situations and behaviors in "*goalie-clear*" process .

is designed to guard when the ball is near. This year, however, the goalie is designed to start the guard motion depending on the distance and speed of the ball and to stand up after its motion. As a result, the goalie does not break the Illegal Defense. In a situation "*play-guard-motion*" the goalie saves with the behavior "*guard_combo*", which is a fusion of the behavior "*guard_down*" and "*guard_up*". In the behavior "*guard_down*" the goalie saves depending on the speed and position of the ball. When an opponent shoots the ball to the left or right side of the goal, the goalie moves left or right and opens the legs to save with the behavior "*guard_down*". After that, in the behavior "*guard_up*" the goalie stands up. Then, there are two ways to stand. If the ball is near the goalie, the goalie pushes the ball with the legs. Otherwise, the goalie stands up quickly. In a situation "*guard-during-approaching*", the goalie saves with the behavior "*approachguard_combo*", as well as the behavior "*guard_combo*", but it is called only when the goalie is approaching the ball and when the opponent kicks the ball.

6.3.4 Clear

The "*clear*" process (cf. Fig. 6.3b) is called only when the ball is near to the goalie or its own goal. In a situation "*approach-ball*", the goalie in front of the goal approaches the ball with the behavior "*approachCatch*". When the goalie is near enough to kick the ball, in a situation "*clear-ball*", the goalie clears it out of a penalty area with the behavior "*shootAfterCatch*", which is the same as our attacker. After clearing it, the "*position*" process may be called, because the goalie may be far away from its own goal. If the goalie could detect an opponent, it can go to clear the ball over the penalty area without fears.

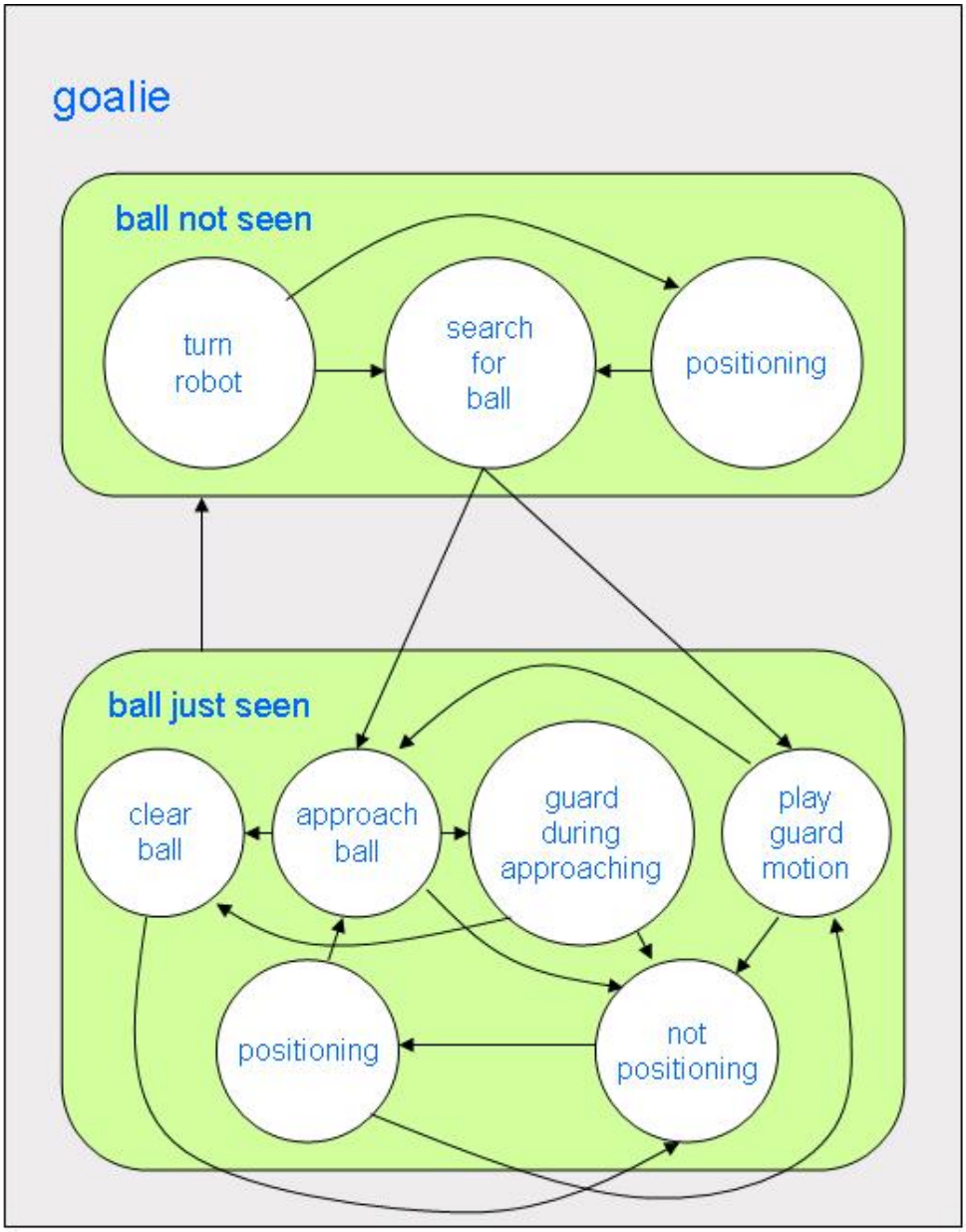


Figure 6.4: Above "*goalie*" expresses the general goal keeper script.

6.3.5 Conclusion

The goalie's script is such as Fig. 6.4. In the "*goalie*", all the processes ("*position*", "*search*", "*guard*", and "*clear*") are included. It is possible to divide them into two groups. One group is "*ball-not-seen*" which consists of situations which the goalie does not see the ball. The other is "*just-ball-seen*" which consists of situations which the goalie sees the ball. In the former, if the goalie is in front of the goal and its situation is "*search-for-ball*", the goalie tries to find the ball with the behavior "*searchBall*". When the goalie finds the ball, its situation changes to the situation "*approach-ball*" or "*play-guard-motion*". In the latter, all the processes except process "*search*" are called depending on the position, the speed of the ball, and so on. When the goalie does not see the ball, the situation of the goalie is changed to the group "*ball-not-seen*" at once.

6.3.6 Problems for Next Year

We must improve our positioning skills. For instance, there is a possibility that the goalie loses its own position by colliding with the edges of its own goal. In order to avoid this trouble, we need to develop a collision detection system by monitoring its joint angles, our positioning system to avoid colliding with the edges, a more robust localization system, and so on. In the "*position*" process, the goalie is in the center of the goal this year. However, we hope that the goalie is designed to position with the information of the ball. For example, the goalie walks a few steps to the side the ball is in, because the goalie more effectively fills up a chink of the goal and saves. In the "*search*" process, the ball sometimes is in a blind spot of the goalie, nevertheless it is in front of its own goal. For example, it is possible that the ball is at the back of the goalie. The nearer to the goal the goalie is, the smaller the blind spot in front of the goal is. Therefore, the behavior is needed to let the goalie step back to its own goal while searching for the ball. The "*guard*" process needs to estimate the speed and direction of the ball more robust. In the "*clear*" process, when the ball is between the goalie and its own goal, it is necessary that the goalie must not shoot toward its own goal but clear the ball out of the penalty area.

Chapter 7

Shot Motions

This year, we prepared the six kinds of shootable areas, i.e. near left(NL), near center(NC), near right(NR), far left(FL), far center(FC), and far right(FR) areas from its view point, in front of a robot as shown in Figure 7.1, so that it can make a precise shot even though it approaches a ball roughly. In each shootable area, the five kinds of shots for left, left oblique, forward, right oblique, and right directions were allotted. In a word, We used the $5 \times 6 = 30$ kinds of shots in all except for shots after catching.

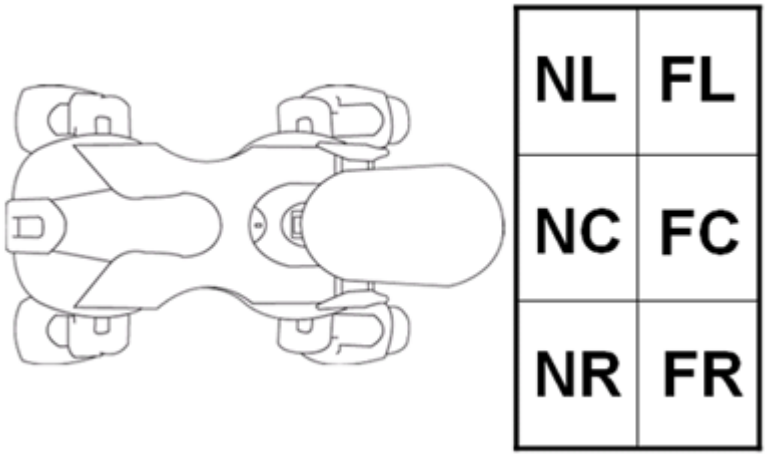


Figure 7.1: The six kinds of shootable areas in front of a robot.

7.1 Shot with its chest

This shot is for the forward direction on the FL, FC, and FR areas. When a robot performs this shot, it throws out its whole body ahead as shown in Figure **??**. This

shot has the advantage of high hitting ratio by using large area of its chest. However, this shot also has the disadvantage of a concentrated load for servo motors of its front legs, because the robot must bear its full weight with its front legs which bent in a L-shape as shown in Figure 7.2(b). We do not want to make robots to perform such a high-loaded motion, because the robots, that is AIBOs, were unfortunately halted in production by Sony. Therefore, we made them shoot a ball after catching it, as described in Section 7.4, whenever possible.

(a) Default pose (b) Chest shot

Figure 7.2: The motion of the shot with its chest.

7.2 Shot with its leg

This shot is for the left, left oblique, right oblique, and right directions on the NL, NC, and NR areas. When a robot performs this shot, it twists its whole body and brandishes its leg as shown in Figure 7.3. That is because AIBOs do not have enough strong servo motors for their joints to kick a ball far away by only brandishing their left or right front leg.

7.3 Shot with its head

This shot is for the following two purposes. One is to move a ball toward the forward direction on the NL, NC, and NR areas, and the other is to move it toward the left, left oblique, right oblique, and right directions on the FL, FC, and FR areas. We define the former type of shot as Type1 and the latter type of shot as Type2.

(a) Expand the front leg which kicks the ball

(b) Stretch its front leg and twist its whole body

(c) End of the shot

Figure 7.3: The motion of the shot with its leg.

7.3.1 Type1 (when the ball is near)

This shot is one of the most difficult shot to make. That is because simply swinging only its head does not cause that the shot can not move the ball far away. The robot must use its whole body for a strong shot. When the robot performs this shot, it put its weight on the ball by raising its body and stretching all the legs as shown in Figure 7.4. The ball runs straightforward because it passes through between its front legs.

The robot tends to slip and moves backward when it stretches the front and rear legs, because its center of gravity is located anteriorly. We must make adjustments to the shot motion so that its legs can catch the field firmly by sticking the pads of its front legs and the claws of its rear legs. If the field has a strong friction and does not slip easily, we must make adjustments so that its front legs can slide on the field, because there is a strong possibility that the pads of its front legs catch the field too much and the servo motors of its legs are damaged.

The robot can frequently lose its balance and fail to hit the ball when it makes this shot in a soccer game, even though we make adjustments firmly by using our motion editor. That is because the shot motion in the adjustment time is different from that in a soccer game. One of the factors causing this difference is that the start joint angles of each leg in walking is different from that in staying. Another is that the shot motion has few processing time in a soccer game, since image processing takes a lot of the calculation time.

(a) Raise body with its legs (b) Stretch its legs and drop down its body

Figure 7.4: The ball is controlled with a stretch of the front legs.

7.3.2 Type2 (when the ball is far)

When the ball is far, it is necessary to move its body forward. We made the shot that the robot thrusts out and swings down its head in the motion of Type1.

7.4 Shot after catch

We made the motion to catch the ball before shots. This motion puts the ball on the sweet spot where the robot can made the most powerful shot.

We used the chest shot and *Abe* shot as shot after catching the ball. The chest shot is almost the same motion described in Sec 7.1 except for not bending its front legs. Therefore, the concentrated load for servo motors of front legs is small. Because the robot only pushes the ball with its chest, this shot motion finishes very fast.

Abe shot can move the ball from the one end line to the other end line of the field. The details are written in our technical report of last year [16].

Chapter 8

Technical Challenges

8.1 The Open Challenge

Our team demonstrated an autonomous learning method in order to acquire ball trapping skills in this year. These skills involve stopping and controlling an oncoming ball and are essential to passing a ball to each other. We first prepared the training equipment shown in Fig. 4.3, and then we showed that one robot can acquire trapping skills on its own and that two robots can acquire them more quickly by sharing their experiences. Our presentation were highly evaluated and took second place in this challenge. The details of our method are given in our paper [17].

8.2 The Passing Challenge

Our robots for the passing challenge were programmed to utilize trapping skills that they practiced and acquired in advance. However, our robots did not work well because we could not quite implement other skills such as dribbling and teammate recognition in time, and so the score of our team was unfortunately zero.

8.3 The New Goal Challenge

This year, in the new goal challenge, we use the same script for the penalty-shootout. We designed the script that the player was able to get a point by only one shot. The strategy of this script is the following.

First, when the player finds the ball, the player tries to approach and catch it. Next, when the player was able to catch the ball, the player turns with holding the ball until seeing the opposite goal ahead. Finally, the player shoots the ball straightforward.

As to recognition of a new goal, we did not especially consider anything. The shape of the goal was not a problem because the player recognized a yellow square as a goal. Our routine can recognize both Fig. 8.1 and Fig. 8.2. It does not become a big difference for our routine.

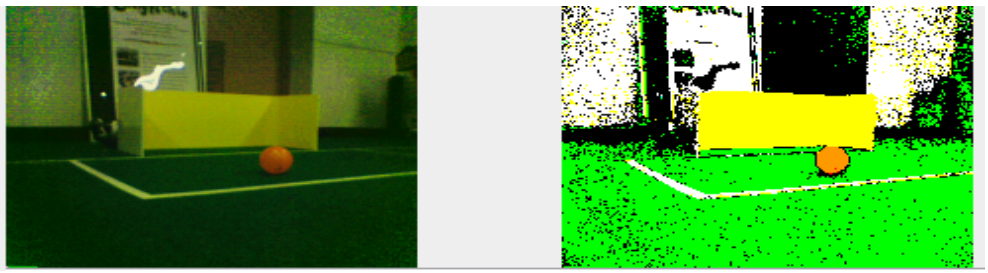


Figure 8.1: The vision of AIBO with normal goal

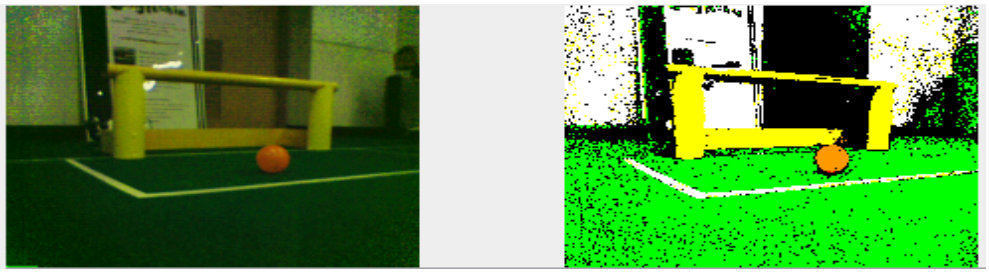


Figure 8.2: The vision of AIBO with new goal

In the practice, the player was able to get points. However in the convention, the shot was not able to reach the goal nevertheless the player was able to shoot. This is because the direction of the shot was not steady and the ball did not reach the goal of the deep turf. In addition, the player was not able to step aside from the opposite player. This is because our player recognition routine was not exact.

Therefore, we were not able to get as much as one point.

Chapter 9

Conclusion

The team Jolly Pochie entered the finals in the RoboCup Japan Four-legged League 2006. In the RoboCup Four-legged League 2006, our team won one game and lost one game in the first round robin pool.

Our mainly improvements of this year are three things. The First is more accurate object recognition of our vision system by improving our object recognition algorithms and our learning tool generating color tables. The second is more robust estimation of ball's location by utilizing a new localization technique. The third is work saving for developing successful ball trapping skills by utilizing an autonomous learning technique for the skills.

We will continue the development of our object recognition routine and the combination of our soccer simulator and on machine learning techniques.

Bibliography

[1] 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.

[2] Aymeric de Cabrol, Patrick Bonnin, Thomas Costis, Vincent Hugel, Pierre Blazevic, and Kamel Bouchefra. A New Video Rate Region Color Segmentation and Classification for Sony Legged RoboCup Application. In *RoboCup 2005*, volume 4020 of *LNAI*, pages 436–443. Springer-Verlag, 2005.

[3] F. Dellaert, D. Fox, W. Burgard, and S. Thrun. Monte Carlo Localization For Mobile Robots. In *Proc. of the IEEE International Conference on Robotics & Automation*, 1998.

[4] M. Dietl, J.-S. Gutmann, and B. Nebel. Cooperative sensing in dynamic environments. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'01)*, 2001.

[5] A. Ferrein, G. Lakemeyer, and L. Hermanns. Comparing Sensor Fusion Techniques for Ball Position Estimation. In *Proc. of the RoboCup 2005 Symposium*, 2005. to appear.

[6] Peggy Fidelman and Peter Stone. Learning ball acquisition on a physical robot. In *2004 International Symposium on Robotics and Automation (ISRA)*, 2004.

[7] D. Fox, W. Burgard, F. Dellaert, and S. Thrun. Monte Carlo Localization: Efficient Position Estimation for Mobile Robots. In *Proc. of the National Conference on Artificial Intelligence*, 1999.

[8] 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.

[9] 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.

[10] C. Hue, J.-P. Le Cadre, and P. Perez. Tracking multiple objects with particle filtering. Technical Report 1361, IRISA, 2000.

[11] J. Inoue, H. Kobayashi, A. Ishino, and A. Shinohara. Jolly Pochie 2005 in the Four Legged Robot League, 2005.

[12] JollyPochie —team for RoboCup soccer 4-legged robot league—. <http://www.i.kyushu-u.ac.jp/JollyPochie/>.

[13] R. Kalman. A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering*, pages 35–45, 1960.

[14] A. Karol and M.-A. Williams. Distributed Sensor Fusion for Object Tracking. In *Proc. of the RoboCup 2005 Symposium*, 2005. to appear.

[15] 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.

[16] Hayato Kobayashi, Tsugutoyo Osaki, Akira Ishino, Jun Inoue, Narumichi Sakai, Satoshi Abe, Shuhei Yanagimachi, Tetsuro Okuyama, Akihiro Kamiya, Kazuyuki Narisawa, and Ayumi Shinohara. Jolly Pochie Technical Report RoboCup2005. Technical report, Jolly Pochie, 2005.

[17] Hayato Kobayashi, Tsugutoyo Osaki, Eric Williams, Akira Ishino, and Ayumi Shinohara. Autonomous learning of ball trapping in the four-legged robot league. In *Proc. RoboCup International Symposium 2006*, LNCS. Springer-Verlag, 2007. to appear.

[18] Nate Kohl and Peter Stone. Machine learning for fast quadrupedal locomotion. In *The Nineteenth National Conference on Artificial Intelligence*, pages 611–616, 2004.

[19] R. Matthew Kretchmar. Parallel reinforcement learning. In *The 6th World Conference on Systemics, Cybernetics, and Informatics.*, 2002.

[20] C. Kwok and D. Fox. Map-based Multiple Model Tracking of a Moving Object. In *Proc. of the RoboCup 2004 Symposium*, 2004.

[21] T. Röfer, T. Laue, and D. Thomas. Particle-Filter-Based Self-Localization Using Landmarks and Directed Lines. In *RoboCup 2005: Robot Soccer World Cup IX, Lecture Notes in Artificial Intelligence, Springer*, 2005. to appear.

[22] D. Schulz, W. Burgard, D. Fox, and A.B. Cremers. Tracking Multiple Moving Objects with a Mobile Robot. In *Proc. of the IEEE Computer Society Conference on Robotics and Automation(ICRA)*, 2001.

[23] Christopher Stanton and Mary-Anne Williams. A Novel and Practical Approach Towards Color Constancy for Mobile Robots Using Overlapping Color Space Signatures. In *RoboCup 2005*, volume 4020 of *LNAI*, pages 444–451. Springer-Verlag, 2005.

[24] Peter Stone, Richard S. Sutton, and Gregory Kuhlmann. Reinforcement learning for robocup soccer keepaway. *Adaptive Behavior*, 13(3):165–188, 2005.

[25] 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.

[26] A. Stroupe, M.-C. Martin, and T. Balch. Distributed Sensor Fusion for Object Position Estimation by Multi-Robot Systems. In *Proc. of the IEEE International Conference on Robotics and Automation*. IEEE, 2001.

[27] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[28] 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.

[29] 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.